IC Compiler Zroute User Guide

Version C-2009.06, June 2009

SYNOPSYS®

Contents

Preface

This preface includes the following sections:

- What's New in This Release
- About This Guide
- Customer Support

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *IC Compiler Release Notes* in SolvNet.

To see the *IC Compiler Release Notes,*

1. Go to the Download Center on SolvNet located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select IC Compiler, and then select a release in the list that appears.

About This Guide

The IC Compiler tool provides a complete netlist-to-GDSII or netlist-to-clock-tree-synthesis design solution, which combines proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementations throughout the design flow.

This guide describes the usage of Zroute in the IC Compiler implementation and integration flow. The flow is described in *IC Compiler Implementation User Guide*, which also describes the usage of the classic router in the flow. For information about defining the routing design rules, see the *IC Compiler Technology File and Routing Rules Reference Manual*.

Audience

This user guide is for design engineers who use Zroute to route designs in the IC Compiler flow.

To use IC Compiler, you need to be skilled in physical design and design synthesis and be familiar with the following:

- Physical design principles
- The Linux or UNIX operating system
- The tool command language (Tcl)

Related Publications

For additional information about IC Compiler, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to refer to the documentation for the following related Synopsys products:

- Design Compiler
- Milkyway Environment

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* `[`*`pin2 ... pinN`*`]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to https://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Introduction to Zroute

Zroute is the next generation Synopsys routing technology. Offering quality-of-results (QoR) improvements, Zroute is better suited for design-for-manufacturing (DFM) and efficiently handles advanced design rules for 45 nm and below technologies. It is architected for multicore hardware and it also works on designs set up with the classic router. Zroute is included with the IC Compiler package and the IC Compiler-PC package, but not in the IC Compiler-XP or IC Compiler-DP package.

Note:

Although using either the classic router or Zroute on your design is preferred, running Zroute on your design after running the classic router is possible. However, after you have run Zroute on your design, you should no longer run the classic router. Doing so results in the following message:

```
Error: It is not recommended to use classic router commands after
Zroute commands. (RT-302)
```

You can check the state of your design by using the `is_zrt_routed_design` command.

Zroute has five routing engines: global routing, track assignment, detail routing, ECO routing, and routing verification. You can invoke global routing, track assignment, and detail routing by using the `route_opt` core command; by using task-specific commands; or by using an autoroute command. You invoke ECO routing and routing verification by using task-specific commands.

During routing, Zroute concurrently optimizes wire length and via count. Every routing step pays close attention to wire length and via count. In addition, you can set options to enable extra steps that specifically target wire length and via count. This optimization, combined with other routing constraints, routes the complete design with less total wire length and a minimum number of vias. You can control the optimization effort level. There are no separate commands to do wire length or via optimization.

Zroute concurrently analyzes and corrects antenna rules, as well as other design rules specified in the Milkyway technology file.

This chapter contains the following sections:

- Zroute Features
- Basic Zroute Flow
- Prerequisites for Zroute

Zroute Features

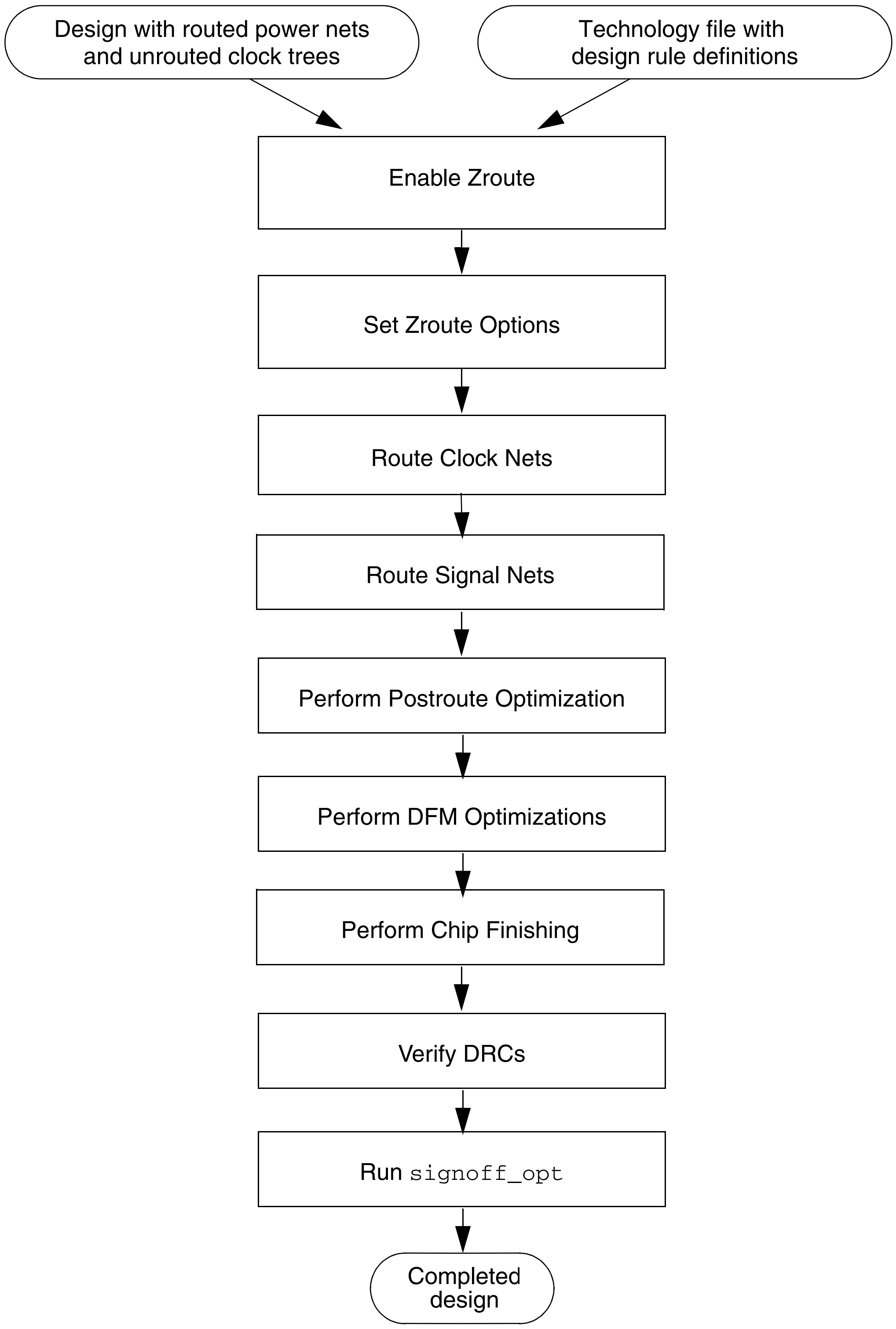
Zroute includes the following main features:

- Multithreading on multicore hardware for all routing steps, including global routing, track assignment, and detail routing
- A realistic connectivity model where Zroute recognizes electrical connectivity as long as the rectangles touch; it does not require the center lines of wires to connect
- A dynamic maze grid that permits Zroute to go off-grid to connect pins, while retaining the speed advantages of gridded routers
- A polygon manager, which allows Zroute to recognize polygons and to understand that design rule checks (DRCs) are aimed at polygons
- Concurrent optimization of design rules, antenna rules, wire optimization, and via optimization during detail routing
- Support for soft rules built into global routing, track assignment, and detail routing
- Timing- and crosstalk-driven global routing, track assignment, detail routing, and ECO routing
- Intelligent design rule handling, including merging of redundant design rule violations and intelligent convergence
- Full integration into the IC Compiler `route_opt` flow
- Net group routing with layer constraints and nondefault routing rules
- Clock routing
- Route verification
- Optimization for DFM and design-for-yield (DFY) using a soft rule approach
- Improved ease of use due to fewer options and commands and no parameters
- Standard messaging for errors and warnings

Basic Zroute Flow

Figure 1-1 shows the basic Zroute flow, which includes clock routing, signal routing, DFM optimizations, and route verification.

Figure 1-1 Basic Zroute Flow


Design with routed power nets and unrouted clock trees
Technology file with design rule definitions
Enable Zroute
Set Zroute Options
Route Clock Nets
Route Signal Nets
Perform Postroute Optimization
Perform DFM Optimizations
Perform Chip Finishing
Verify DRCs
Run signoff_opt
Completed design

Prerequisites for Zroute

Before running Zroute, you must complete power routing and clock tree synthesis but not clock tree routing on your design. In addition, all design rules must be defined in the Milkyway technology file. Unlike the classic router, Zroute gets all of the design rule information from the technology file, instead of a combination of parameters and the technology file. For more information about the technology file and defining routing design rules, see the *IC Compiler Technology File and Routing Rules Reference Manual*.

Before you can run Zroute, you must ensure that the design and physical library meet the following requirements:

- Design requirements

  Before starting signal routing with Zroute, the power nets must be routed by using the existing prerouter. Using the prerouter to route power and ground nets does not change when using Zroute. In addition, the clock trees must by synthesized but not routed.

  You can use the prerouter to preroute signal nets before running Zroute. The prerouted signal nets are marked as user nets. By default, Zroute does not rip-up and reroute these user nets. Zroute makes only minor changes to correct DRC violations.

- Library requirements

  Zroute uses the same Milkyway design library as the classic router, including the technology file and the standard cell and macro cell routing (FRAM) views generated by blockage, pin, and via (BPV) extraction.

  Important:
  Unlike the classic router, which uses both default vias and vias that were used during BPV extraction for nonpin access, Zroute uses only default vias for nonpin access. Make sure that all vias that you want Zroute to use are defined as default vias (`isDefaultContact` attribute is 1) in the technology file.

2

Setting Up for Zroute

This chapter describes the setup tasks you must complete before running Zroute commands. It contains the following sections:

Enabling Zroute

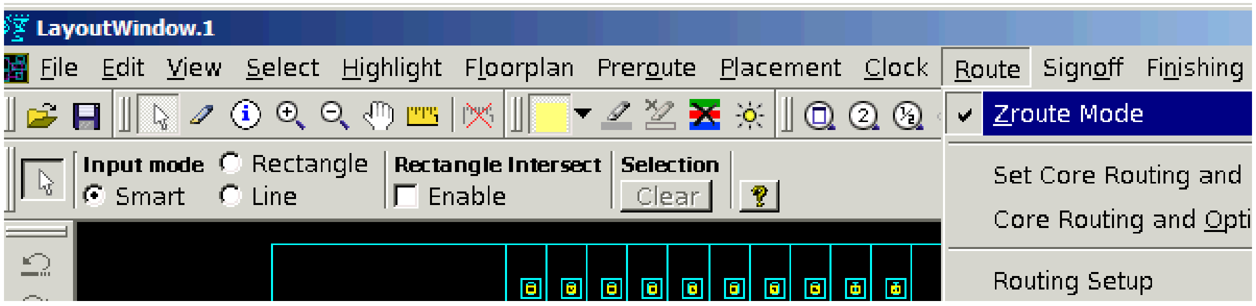
To enable Zroute, set the Zroute route mode option to true. The setting for the Zroute mode option is saved in the Milkyway design database.

If you are using the command line, enter the following command:

```
icc_shell> set_route_mode_options -zroute true
```

If you are using the GUI, select Zroute Mode in either the Route or Finishing menu, as shown in Figure 2-1.

Figure 2-1 Enabling Zroute in the GUI


LayoutWindow.1
File Edit View Select Highlight Floorplan Preroute Placement Clock Route Signoff Finishing
Input mode Rectangle Smart Line
Rectangle Intersect Enable
Selection Clear
✓ Zroute Mode
Set Core Routing and
Core Routing and Opti
Routing Setup

When you enable Zroute, the following commands use Zroute instead of the classic router:

- `report_congestion`
- `clock_opt`
- `route_opt`
- `optimize_clock_tree` (when run on a postroute design)
- `signoff_opt`

Note:
In addition to these commands that can use either Zroute or the classic router, there are several Zroute-specific commands. These commands are identified by the *zrt* string in the command name. Use the `help *zrt*` command to identify these Zroute-specific commands.

Enabling Multithreading

Zroute is designed as a multithreaded router. Unlike distributed routing, which runs on multiple machines, multithreading uses multiple CPUs or multiple cores on the same machine, using a single process memory image. During multithreaded routing, Zroute performs routing tasks in parallel. Each of these tasks is performed by a separate thread. The nature of the parallel algorithm might change depending on the routing task being performed. In comparison, distributed routing partitions the design physically into several partitions and assigns the partitions to separate processes, each of which executes the same algorithm. When the processes complete, the errors at the partition boundaries must be resolved. Zroute supports only multithreading; it does not support distributed routing.

When you run routing with a single thread, the result is deterministic; if you start with the same design, you will always get the same result. However, if you use multiple threads, the routing result will not be deterministic; the QoR might be slightly different between runs. With multiple threads, the end result depends on which thread processes which partition and the order in which the threads process the partitions.

If you have multiple cores on your machine, enable multithreading to reduce the elapsed time required to perform routing. By default, multithreading is not enabled. To enable it, set the `set_host_options -max_cores` option to a value greater than one and less than or equal to the number of cores available on your machine. You need one IC Compiler license for every four threads. For example, to run eight threads, you need two IC Compiler licenses. For more information about the `set_host_options` command, see the man page.

Note:
When you enable multithreading by using the `set_host_options` command, it enables multithreading for all IC Compiler commands that support multithreading, not just Zroute.

When you enable multithreading, Zroute creates and uses the specified number of threads, even if the number is more than the number of available cores. The router does not know how many cores the machine has or its shared CPU usage. You must set an appropriate number of threads, so that the router does not try to use more resources than it has. Overthreading can reduce the routing performance because the extra threads compete for resources.

For best performance, do not run more than one thread per available core, where the number of available cores is the number of CPUs times the number of cores per CPU.

For example, if your machine has 2 CPUs and each CPU has 3 cores, specify 6 as the maximum number of threads:

```
icc_shell> set_host_options -max_cores 6
```

Specifying Route Guides

You can create or remove route guides that do the following for a specific area of your design:

- Prevent routing for signal or prerouted nets
- Change the wiring direction
- Control wiring density
- Fix violations

To create a route guide, use the `create_route_guide` command (or choose Floorplan > Create Route Guide in the GUI). Table 2-1 defines the `create_route_guide` options. For more information about these options, see the man page.

Table 2-1 create_route_guide Command Options

Command option	GUI object	Description
`-name` *`name`*	Name box	Specifies the name of the route guide.
`-repair_as_single_sbox`	"Repair as single SBox" check box	Enables fixing of violations.
`-no_signal_layers` *`layers`*	"No signal route on layers" check box and list	Prevents the routing of signal wires on the specified layers.
`-zero_min_spacing`	"Zero min-spacing" check box	Allows the routing of wires within the keepout margin of the route guide.
`-no_preroute_layers` *`layers`*	"No automatic preroutes on layers" check box and list	Prevents automatic preroutes on the specified layers.
`-preferred_direction_only_layers` *`layers`*	"Layers with preferred direction only" check box and list	Specifies the layers that must be routed in the preferred direction.
`-horizontal_track_utilization` *`percentage`*	Horizontal box in "Route track utilization" section	Specifies the allowable horizontal track utilization.
`-vertical_track_utilization` *`percentage`*	Vertical box in "Route track utilization" section	Specifies the allowable vertical track utilization.

Table 2-1 create_route_guide Command Options (Continued)

Command option	GUI object	Description
`-coordinate` *`rectangle`*	Coordinates box	Specifies the coordinates for the route guide. In the GUI, you can type the coordinates in the dialog box or draw the rectangle in the layout view. You can also specify that the route guide should snap to the minimum grid, placement site, routing track (the default), middle routing track, or user grid. From the command line, the snapping is controlled by the `set_object_snap_type` command.
`-switch_preferred_direction`	"Switch preferred direction" check box	Switches the preferred wiring direction.

To find route guides, use the `get_route_guides` command. For example, to get all the route guides in your design, enter

```
icc_shell> get_route_guides *
```

You can also find route guides by using a filter expression (`-filter` *`expression`*) or by using rectangular areas that encompass (`-within {{`*`x1 y1`*`} {`*`x2 y2`*`}}`) or touch (`-touch {{`*`x1 y1`*`} {`*`x2 y2`*`}}`) the guides.

To remove route guides, use the `remove_route_guides` command. You can remove a collection of route guides, such as that returned by the `get_route_guides` command; all route guides (`-all`); or a specific named route guide (`-name`).

For more information about the `get_route_guides` or `remove_route_guide` command, see its man page.

Defining Routing Blockages

A routing blockage defines a region where no routing is allowed on a specific layer. You define routing blockages by using the `create_routing_blockage` command. To define a routing blockage, you must specify

- The blockage layers to which the routing blockage applies

  Use the `-layers` option to specify the blockage layers. Valid values for the blockage layers are metal1Blockage-metal15Blockage, via1Blockage-via14Blockage, polyBlockage, and polyContBlockage. To query the Milkyway design library for the blockage layers, use the `get_layers -include_system` command.

  You can specify one or more blockage layers. If the routing blockage is defined on a metal blockage layer, it is a metal routing blockage. If the routing blockage is defined on a via blockage layer, it is a via routing blockage.

- The region of the blockage

  Use the `-bbox` option to specify the region for a rectangular routing blockage. Use the `-boundary` option to specify the region for a rectilinear routing blockage.

For example, to create rectangular routing blockages on the metal1 and via1 blockage layers, enter the following command:

```
icc_shell> create_routing_blockage \
-layers {metal1Blockage via1Blockage} -bbox {x1 y1 x2 y2}
```

When IC Compiler creates routing blockages, it assigns a name of RB_*id* to each routing blockage.

To find routing blockages, use the `get_routing_blockages` command. For example, to get all the routing blockages in your design, enter

```
icc_shell> get_routing_blockages *
```

You can also find routing blockages by using a filter expression (`-filter` *`expression`*) or by using rectangular areas that encompass (`-within {{`*`x1 y1`*`} {`*`x2 y2`*`}}`) or touch (`-touch {{`*`x1 y1`*`} {`*`x2 y2`*`}}`) the blockages.

To remove routing blockages, use the `remove_routing_blockage` command.

For more information about the `get_routing_blockages` or `remove_routing_blockage` command, see its man page.

Controlling Pin Connections

By default, Zroute connects a signal route to a pin by using wires or vias anywhere on the pin. You can restrict the allowed types of pin connections on a per-layer basis by setting the `connect_within_pins` common Zroute option.

Valid values for this option are

- `off` (the default)

  There are no restrictions on pin connections.

- `via_standard_cells_pins`

  Only the connections to standard cell pins by using a via are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape.

  There are no restrictions on signal routes connected to macro cell and pad cell pins by using a via or to any pins by using wires.

- `via_wire_standard_cell_pins`

  The connections to standard cell pins by using a via or a wire are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape. When using a wire, the wire must be contained within the pin shape.

- `via_all_pins`

  The connections to any pins (standard cell, macro cell, or pad cell) by using a via are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape.

  There are no restrictions on signal routes connected to any pins by using wires.

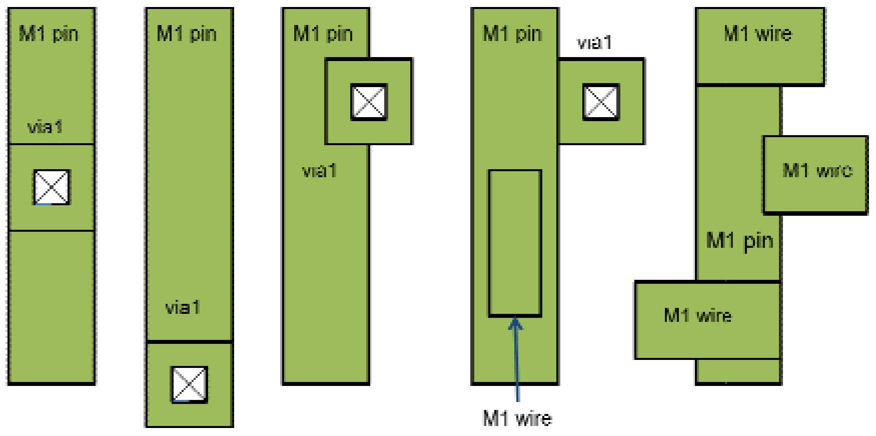
- `via_wire_all_pins`

  The connections to any pins by using a via or a wire are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape. When using a wire, the wire must be contained within the pin shape.

For example, if you enter the following command (or use the default settings), all of the connections shown in Figure 2-2 are valid and no DRC violations are reported:

```
icc_shell> set_route_zrt_common_options -connect_within_pins {{m1 off}}
```

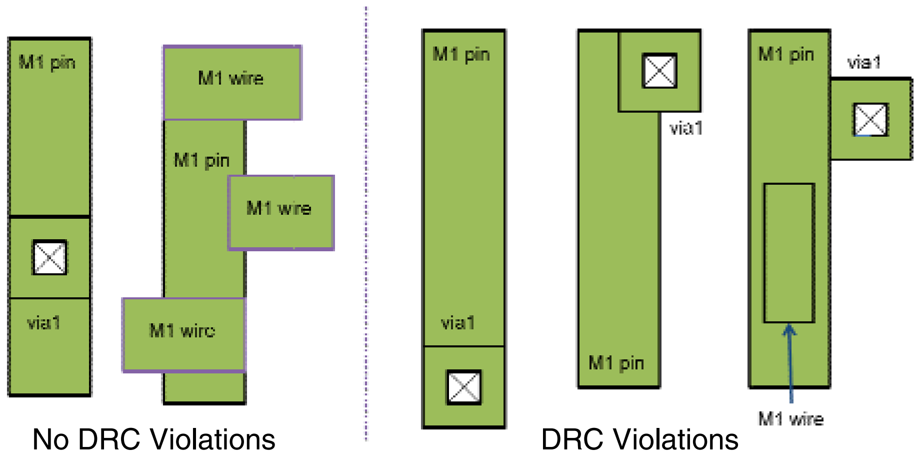
Figure 2-2 Unrestricted Pin Connections



If you set the mode for metal1 to `via_all_pins`, as shown in the following example, the via enclosures must be inside the pin shape. The connections shown on the left side of Figure 2-3 are valid; however, the connections on the right side of the figure cause DRC violations.

```
icc_shell> set_route_zrt_common_options \
-connect_within_pins {{m1 via_all_pins}}
```

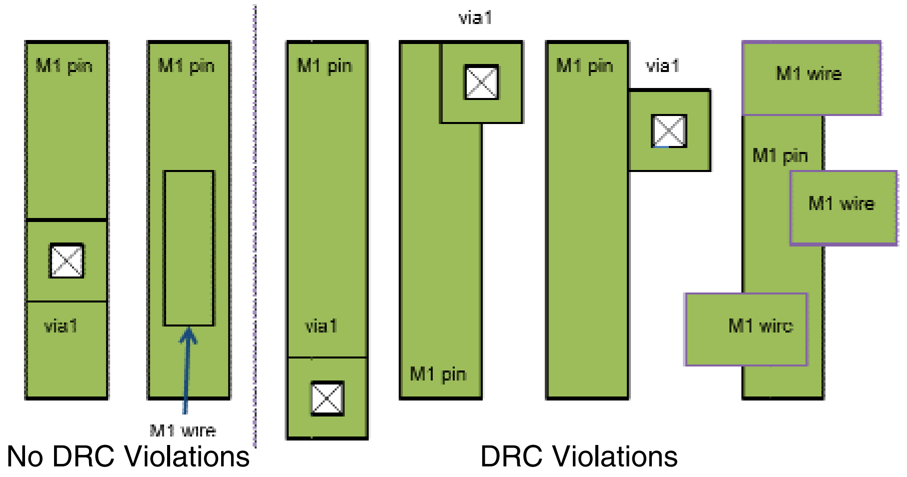
Figure 2-3 Restricted Via-to-Pin Connections



If you set the mode for metal1 to `via_wire_standard_cell_pins`, as shown in the following example, both the via enclosures and wires must be inside the pin shape. The connections shown on the left side of Figure 2-4 are valid; however the connections on the right side of the figure cause DRC violations.

```
icc_shell> set_route_zrt_common_options \
-connect_within_pins {{m1 via_wire_standard_cell_pins}}
```

Figure 2-4 Restricted Via-to-Pin and Wire-to-Pin Connections



Using Nondefault Routing Rules

Zroute supports the use of nondefault routing rules, both for routing and for shielding. Before you can use a nondefault routing rule, you must define it. The following sections describe how to define and apply nondefault routing rules.

Defining Nondefault Routing Rules

You define nondefault or default routing rules for specific nets by using the `define_routing_rule` command (or by choosing Route > Routing Setup > Define Routing Rule in the GUI). These rules define wire width and spacing rules and via types. Table 2-2 defines the `define_routing_rule` options. For more information about these options, see the man page.

Table 2-2 define_routing_rule Command Options

Command option	**GUI object**	**Description**
`rule_name`	"New rule name" box	Specifies the name of the rule. This argument is required.
	"Reference rule" section	Specifies the name of the reference rule. This option is required.
`-reference_rule_name`	Specified radio button	
`-default_reference_rule`	Default radio button	

Table 2-2 define_routing_rule Command Options (Continued)

Command option	GUI object	Description
`-taper_level`	"Taper level" box	Specifies the tapering level. The default tapering level is 1.
`-snap_to_track`	"Snap to track" check box	Snaps shielding wires to the track when selected. By default, snapping is not enabled.
`-multiplier_width`	"Multiplier width" box	Specifies the layer width multiplier.
`-multiplier_spacing`	"Multiplier spacing" box	Specifies the layer spacing multiplier.
	Metal table section	Specifies the width and spacing rules.
`-widths`	Width column	
`-spacings`	Spacing column	
`-shield_widths`	"Shield width" column	
`-shield_spacings`	"Shield spacing" column	
`-via_cuts`	"Contact table" section	Specifies the via types.

Applying Nondefault Routing Rules

IC Compiler provides two commands for assign nondefault routing rules to nets:

- `set_clock_tree_options`

  This command assigns nondefault routing rules to clock nets before clock tree synthesis.

- `set_net_routing_rule`

  This command assigns nondefault routing rules to signal nets and to clock nets after clock tree synthesis.

Applying Nondefault Routing Rules to Clock Nets

To assign a nondefault routing rule to clock nets, use the `-routing_rule` option of the `set_clock_tree_options` command.

Note:
This command works only before clock tree synthesis. If the clock tree has already been synthesized, use the `set_net_routing_rule` command to apply the nondefault routing rules, as described in "Applying Nondefault Routing Rules to Signal Nets" on page 2-11.

For example, to assign a shielding rule called shield_rule to the nets in the CLK clock tree before clock tree synthesis, enter the following command:

```
icc_shell> set_clock_tree_options -clock_tree CLK \
-routing_rule shield_rule
```

By default, the shielding rule applies to all nets in the clock tree. To use the default routing rule for the nets closest to the sink pin, use the `-use_default_routing_for_sinks` option. The default routing rules are used for the specified number of clock tree levels closest to the clock sink. For more information about the `set_clock_tree_options` command, see Chapter 7, "Clock Tree Synthesis," in the *IC Compiler Implementation User Guide*.

Applying Nondefault Routing Rules to Signal Nets

To apply a nondefault routing rule to one or more nets, use the `set_net_routing_rule` command (or choose Route > Routing Setup > Set Net Routing Rule in the GUI). Table 2-3 defines the `set_net_routing_rule` options. For more information about these options, see the man page.

Table 2-3 set_net_routing_rule Command Options

Command option	GUI object	Description
`list_of_nets`	Nets box	Specifies the nets to which to apply the rule. This argument is required.
`-rule` *`rule_name`*	"Routing rule" box	Specifies the name of the nondefault rule to apply. This option is required.
`-top_layer_probe AnyPort | OutPort | AllPort`	"Top layer probe mode" box	Specifies which ports to use for top-layer probing. The default is `AnyPort`.
`-reroute normal | minorchange | freeze`	"Net re-routability" box	Specifies when to reroute nets. The default is `normal`.
`-timing_driven_spacing`	"Timing driven spacing" check box	When timing-driven spacing is enabled, Zroute adds space between critical nets and nets that are parallel to them to reduce the intralayer coupling capacitances. By default, timing-driven spacing is not enabled.

Specifying the Routing Layers

IC Compiler lets you specify which layers can be ignored for routing and which layers are to be ignored for RC and congestion estimation. You can also specify the routing layers to use for specific nets (net layer constraints).

By default, when you set a maximum routing layer, it is a hard constraint. You can change it to a soft constraint or you can allow the use of higher layers only for pin connections by setting the `max_layer_mode` common route option. This option applies to both ignored layers (`set_ignored_layers` command) and net layer constraints (`set_net_routing_layer_constraints` command).

By default, when you set a minimum routing layer, it is a soft constraint. You can change it to a hard constraint or you can allow the use of lower layers only for pin connections by setting the `min_layer_mode` common route option. This option applies to both to ignored layers (`set_ignored_layers` command) and net layer constraints (`set_net_routing_layer_constraints` command).

Specifying the Ignored Layers

To specify the ignored layers, use the `set_ignored_layers` command (or choose Route > Routing Setup > Set Ignored Layers in the GUI).

Table 2-4 shows the `set_ignored_layers` options. For more information about these options, see the man page.

Table 2-4 set_ignored_layers Command Options

Command option	GUI object	Description
`-min_routing_layer` *name*	"Minimum routing layer" check box and list	Specifies the lowest routing layer. Zroute uses the specified layer and the layers above it for routing.
`-max_routing_layer` *name*	"Maximum routing layer" check box and list	Specifies the highest routing layer. Zroute uses the specified layer and the layers below it for routing.
`-rc_congestion_ignored_ layers` *names*	"Select layers to be ignored in RC computation" section	Specifies the layers to ignore for RC and congestion estimation. By default, RC and congestion estimation use the layers you specify for routing.

For example, to define layers M2 through M7 for routing, use the following command:

```
icc_shell> set_ignored_layers -min_routing_layer M2 -max_routing_layer M7
```

To define the same layers for routing and to force the RC and congestion estimation functions to ignore the M1, M2, M8, and M9 layers, use the following command:

```
icc_shell> set_ignored_layers \
   -min_routing_layer M2 -max_routing_layer M7 \
   -rc_congestion_ignored_layers {M1 M2 M8 M9}
```

Specifying Routing Layers for Specific Nets

To specify the routing layers for specific nets, use the `set_net_routing_layer_constraints` command (or choose Route > Routing Setup > Set Net Layer Constraints in the GUI).

Table 2-5 shows the `set_net_routing_layer_constraints` options. For more information about these options, see the man page.

Table 2-5 set_net_routing_layer_constraints Command Options

Command option	GUI object	Description
nets	“Specified nets” box	Specifies the nets to which to apply the routing constraints. This argument is required.
`-min_layer_name` *name*	“Minimum routing layer” list	Specifies the lowest routing layer. Zroute uses the specified layer and the layers above it for routing the specified nets.
`-max_layer_name` *name*	“Maximum routing layer” list	Specifies the highest routing layer. Zroute uses the specified layer and the layers below it for routing the specified nets.

For example, to define layers M2 through M7 for routing net n1, use the following command:

```
icc_shell> set_net_routing_layer_constraints \
-min_layer_name M2 -max_layer_name M7 [get_nets n1]
```

Setting Zroute Options

You can specify global routing options, track assignment options, detail routing options, and common routing options (options that affect all three routing engines). Zroute uses these settings whenever you perform routing functions. When you run a routing command, Zroute writes the settings for any routing options that you have set (or that the tool has set for you) in the routing log. To display the settings for all routing options, not only those that have been set, set the `verbose_level` common Zroute option to 1.

```
icc_shell> set_route_zrt_common_options -verbose_level 1
```

Setting Common Route Options

Common route options are used to define routing options that affect global routing, track assignment, and detail routing. When you save the design in Milkyway format, the common route option settings are saved with the design.

- To set the common route options, use the `set_route_zrt_common_options` command (or choose Route > Routing Setup > Set Common Route Options in the GUI). To reset the options to their default values, use `set_route_zrt_common_options -default true` (or click Default in the GUI).
- To report the settings of all common route options, use the `report_route_zrt_common_options` command.
- To return the value of a specific common route option, use the `get_route_zrt_common_options` command.

Table 2-6 lists the Zroute common route options.

Table 2-6 Common Route Options

Option	Valid values	Description
Run control options		
`enforce_voltage_areas`	`off | strict | relaxed`	Specifies the level of enforcement for voltage area constraints. The default is `relaxed`.
`reroute_clock_shapes`	`true | false`	Specifies whether the router can reroute fixed clock wires and vias. The default is `false`.

Table 2-6 Common Route Options (Continued)

Option	Valid values	Description
`reroute_user_shapes`	`true | false`	Specifies whether the router can reroute user-created wires and vias. The default is `false`.
`plan_group_aware`	`off | all_routing | top_level_routing_only`	Specifies whether the plan group constraints are obeyed. The default is `off`.
`child_process_net_threshold`	*`int`* (must be between -1 and 2147483647)	Threshold of number of nets to trigger a separate router process. The default is -1, which means a separate process is never triggered.
`verbose_level`	0 | 1	Sets the verbosity level for the routing log file. The default is 0.
Boundary and blockage options		
`read_user_metal_blockage_layer`	`true | false`	Specifies whether shapes on metal blockage layers should be honored (true) or ignored (false). The default is `false`.
`wide_macro_pin_as_fat_wire`	`true | false`	Specifies whether or not the wide macro or pad pin should have an implicit depth same as its width. The default is `false`.
`route_top_boundary_mode`	`stay_half_min_space_inside | stay_inside | ignore`	Controls the routing behavior near the top boundary. The default is `stay_inside`.
`standard_cell_blockage_as_thin`	`true | false`	Specifies whether or not to treat wide blockages in standard cells as thin wires. The default is `false`.

Table 2-6 Common Route Options (Continued)

Option	Valid values	Description
Layer options		
`freeze_layer`	`{{`*`layer`*` true | false}...}`	Controls whether routing layers are frozen to prevent them from changing during routing. By default (`false`), routing layers are not frozen.
`freeze_via_to_frozen_layer`	`true | false`	Controls the treatment of vias that touch a frozen layer on one side. The default is `false`.
`max_layer_mode`	`soft | allow_pin_connection | hard`	Specifies the strength of the maximum layer constraint. The default is `hard`.
`min_layer_mode`	`soft | allow_pin_connection | hard`	Specifies the strength of the minimum layer constraint. The default is `soft`.
`extra_preferred_direction_wire_cost_multiplier`	`{{`*`layer int`*`}...}`	Specifies the cost multiplier for routing in the preferred direction.
`extra_nonpreferred_direction_wire_cost_multiplier`	`{{`*`layer int`*`}...}`	Specifies the cost multiplier for routing in the nonpreferred direction.
`extra_via_cost_multiplier`	`{{`*`layer int`*`}...}`	Specifies the cost multiplier for vias.

Table 2-6 Common Route Options (Continued)

Option	Valid values	Description
Via options		
`number_of_vias_over_max_layer`	*`int`* (must be between 0 and 15)	Specifies the maximum level of stacked vias that can be used to access pins or fixed routes above the maximum routing layer. The default is 1.
`number_of_vias_under_min_layer`	*`int`* (must be between 0 and 15)	Specifies the maximum level of stacked vias that can be used to access pins or fixed routes below the minimum routing layer. The default is 1.
`via_array_mode`	`off | swap | rotate | all`	The default is `all`.
`post_detail_route_redundant_via_insertion`	`off | low | medium | high`	Enables automatic redundant via insertion after detail routing and ECO routing. The default is `off`.
`concurrent_redundant_via_mode`	`off | reserve_space | insert_at_high_cost`	Enables concurrent redundant via insertion during initial routing. The default is `off`.
`concurrent_redundant_via_effort_level`	`low | medium | high`	Specifies the effort level for concurrent redundant via insertion during initial routing. The default is `low`.
`eco_route_concurrent_redundant_via_mode`	`off | reserve_space`	Enables concurrent redundant via insertion during ECO routing. The default is `off`.
`eco_route_concurrent_redundant_via_effort_level`	`low | medium | high`	Specifies the effort level for concurrent redundant via insertion during ECO routing. The default is `low`.
`rotate_default_vias`	`true | false`	The default is `true`.

Table 2-6 Common Route Options (Continued)

Option	Valid values	Description
Miscellaneous options		
`off_grid_routing_mode`	`allow_off_grid | allow_end_points_ off_grid | no_off_grid`	Controls when off-grid routing can be used. The default is `allow_off_grid`.
`single_connection_to_pins`	`off | standard_cell_pins | all_pins`	The default is `off`, except for nets that use a nondefault width routing rule. For those nets, the value is always `all_pins`.
`mark_clock_nets_minor_change`	`true | false`	Specifies whether only minor changes can be made to clock nets. The default is `true`.
`route_soft_rule_effort_level`	`off | min | low | med | high`	Specifies the effort level to rip up and reroute soft rule design rule constraints (DRCs). The default is `medium`.
`threshold_noise_ratio`	*`float`* (must be between 0 and 1)	Specifies the noise threshold as a fraction of the operating voltage. The default is 0.35.
`track_auto_fill`	`true | false`	Specifies whether or not to fill empty space with routing tracks. The default is `true`.
`connect_within_pins`	`{{`*`layer`*` off | via_standard_ cell_pins | via_wire_standard_ cell_pins | via_all_pins | via_wire_all_ pins}...}`	Specifies which pin connections must be made within the pins for the specified layer. By default, this option is `off` for all layers.

For more information about the `set_route_zrt_common_options` command, see the man page.

Setting Global Route Options

Global route options are used to define routing options that affect global routing. When you save the design in Milkyway format, the global route option settings are saved with the design.

- To set the global route options, use the `set_route_zrt_global_options` command (or choose Route > Routing Setup > Set Global Route Options in the GUI). To reset the options to their default values, use `set_route_zrt_global_options -default true` (or click Default in the GUI).
- To report the settings of all global route options, use the `report_route_zrt_global_options` command.
- To return the value of a specific global route option, use the `get_route_zrt_global_options` command.

Table 2-7 lists the Zroute global route options.

Table 2-7 Global Route Options

Option	Valid values	Description
Run control options		
`timing_driven`	`true | false`	Enables (true) or disables (false) timing-driven global routing. The default is `false`.
`crosstalk_driven`	`true | false`	Enables (true) or disables (false) crosstalk-driven global routing. The default is `false`.
`congestion_map_only`	`true | false`	If false (the default), create both glinks and a congestion map. If true, create the congestion map without creating glinks.
`effort`	`minimum | low | medium | high`	Specifies the global route effort. The default is `medium`.

Table 2-7 Global Route Options (Continued)

Option	Valid values	Description
Clock net options		
`clock_topology`	`comb` | `normal`	Specifies the global-routing clock topology. The default is `normal`.
`comb_distance`	*`int`* (must be between 0 and 50)	Specifies the number of global route cells within which to connect clock pins to the clock mesh. The default is 2.
`comb_max_connections`	*`int`* (must be between -1 and 50)	Specifies the maximum number of pins that can be connected to any clock strap. The default is -1, which means there is no maximum.
Macro options		
`macro_boundary_track_utilization`	*`int`* (must be between 0 and 100)	Specifies the percentage of tracks to be used along macro boundaries. The default is 100.
`macro_boundary_width`	*`int`* (must be between 0 and 10)	Specifies the width of the macro boundary in terms of global route cells. The default is 5.
`macro_corner_track_utilization`	*`int`* (must be between 0 and 100)	Specifies the percentage of tracks to be used along macro corners. The default is 100.

For more information about the `set_route_zrt_global_options` command, see the man page.

Setting Track Assignment Options

Track assignment options are used to define routing options that affect track assignment. When you save the design in Milkyway format, the track assignment option settings are saved with the design.

- To set the track assignment options, use the `set_route_zrt_track_options` command (or choose Route > Routing Setup > Set Track Assign Options in the GUI). To reset the options to their default values, use `set_route_zrt_track_options -default true` (or click Default in the GUI).
- To report the settings of all track assignment options, use the `report_route_zrt_track_options` command.
- To return the value of a specific track assignment option, use the `get_route_zrt_track_options` command.

Table 2-8 lists the Zroute track assignment options.

Table 2-8 Track Assignment Options

Option	Valid values	Description
`crosstalk_driven`	`true | false`	Enables (true) or disables (false) crosstalk-driven track assignment. The default is `false`.
`timing_driven`	`true | false`	Enables (true) or disables (false) timing-driven track assignment. The default is `false`.

For more information about the `set_route_zrt_track_options` command, see the man page.

Setting Detail Route Options

Detail route options are used to define routing options that affect detail routing. When you save the design in Milkyway format, the detail route option settings are saved with the design.

- To set the detail route options, use the `set_route_zrt_detail_options` command (or choose Route > Routing Setup > Set Detail Route Options in the GUI). To reset the options to their default values, use `set_route_zrt_detail_options -default true` (or click Default in the GUI).

- To report the settings of all detail route options, use the `report_route_zrt_detail_options` command.
- To return the value of a specific detail route option, use the `get_route_zrt_detail_option` command.

Table 2-9 lists the Zroute detail route options.

Table 2-9 Detail Route Options

Option	Valid values	Description
Run control options		
`timing_driven`	`true | false`	Enables (true) or disables (false) timing-driven routing. The default is `false`.
`optimize_wire_via_effort_level`	`off | low | medium | high`	Specifies the effort level used to optimize wire length and via counts. The default is `low`.
`optimize_tie_off_effort_level`	`off | low | high`	Specifies the effort level used to optimize wire length and via counts for tie-off nets. The default is `low`.
`generate_extra_off_grid_pin_tracks`	`true | false`	Specifies whether to generate extra off-grid routing tracks for pin connections. The default is `false`.
`generate_off_grid_feed_through_tracks`	`off | low | medium | high`	Specifies the effort level used to generate extra off-grid routing tracks for feedthrough nets between blockages. The default is `low`.
`save_after_iterations`	*`list_of_ints`*	Specifies the iterations after which to save the design. The default is {}.
`save_cell_prefix`	*`string`*	Specifies the prefix to use for intermediate saved designs. The default is *DR*.

Table 2-9 Detail Route Options (Continued)

Option	Valid values	Description
`force_max_number_iterations`	`true | false`	Controls whether the maximum number of iterations must be run when DRCs do not converge. The default is `false`.
`user_defined_partition`	{*llx lly urx ury*}	Specifies the coordinates, in database units, of a rectangular partition to be added for routing.
`cpu_limit`	*int* (must be between -1 and 2147483647)	Specifies the limit for CPU time in minutes. The default is -1, which means there is no limit.
Antenna options		
`antenna`	`true | false`	Enables (true) or disables (false) antenna analysis. The default is `true`.
`antenna_on_iteration`	*int* (must be between 1 and 19)	Specifies the routing iteration at which antenna analysis and optimization is turned on. The default is 1.
`check_antenna_on_pg`	`true | false`	Enables (true) or disables (false) analysis and optimization of antennas on power and ground nets. The default is `false`.
`default_diode_protection`	*float* (must be between 0.00 and 1e+06)	Specifies the diode protection value used for pins during antenna analysis if the value is not specified in the cell. The default is 0.00.
`default_gate_size`	*float* (must be between 0.00 and 1e+06)	Specifies the gate size used for pins during antenna analysis if the gate size is not specified in the cell. The default is 0.00.

Table 2-9 Detail Route Options (Continued)

Option	Valid values	Description
`default_port_external_antenna_area`	*float* (must be between 0.00 and 1e+06)	Specifies the antenna area value used for ports (top-level pins) during antenna analysis if the antenna area is not specified in the cell. The default is 0.00.
`default_port_external_gate_size`	*float* (must be between 0.00 and 1e+06)	Specifies the gate size used for ports (top-level pins) during antenna analysis if the gate size is not specified in the cell. The default is 0.00.
`diode_libcell_names`	*list_of_libcells*	Specifies the diode library cells to use for antenna fixing. By default, Zroute selects the diode cells from the library.
`insert_diodes_during_routing`	`true | false`	Specifies whether the router can use diode insertion to fix antenna violations. The default is `false`.
`reuse_filler_locations_for_diodes`	`true | false`	Specifies whether to reuse filler cell locations for inserting diodes. The default is `true`.
`max_antenna_pin_count`	*int* (must be between -1 and 1000000)	Specifies the maximum number of pins on a net on which antenna checking is performed. The default is -1, which means there is no limit.
`merge_gates_for_antenna`	`true | false`	Enables (true) or disables (false) merging gates for antenna analysis. The default is `true`.
`port_antenna_mode`	`float | jump | top_layer`	Specifies how the ports (top-level pins) are treated for antenna consideration. The default is `float`.

Table 2-9 Detail Route Options (Continued)

Option	Valid values	Description
`top_layer_antenna_fix_threshold`	*int* (must be between -1 and 10)	Specifies the threshold for fixing antenna violations on the top routing layer. The default is -1, which means there is no limit.
`antenna_verbose_level`	`0 | 1`	Sets the antenna checking verbosity level in the routing log file. The default is 1.
Pin and port options		
`check_pin_min_area_min_length`	`true | false`	Enables (true) or disables (false) checking for minimum area and minimum length rules on pins. The default is `false`.
`check_port_min_area_min_length`	`true | false`	Enables (true) or disables (false) checking for minimum area and minimum length rules on ports. The default is `true`.
`pin_taper_mode`	`default_width | pin_width | off`	Specifies the pin tapering mode. The default is `default_width`.
`use_wide_wire_to_input_pin`	`true | false`	Specifies whether pin tapering to input pins is disallowed. The default is `false`.
`use_wide_wire_to_macro_pin`	`true | false`	Specifies whether pin tapering to macro pins is disallowed. The default is `false`.
`use_wide_wire_to_output_pin`	`true | false`	Specifies whether pin tapering to output pins is disallowed. The default is `false`.

Table 2-9 Detail Route Options (Continued)

Option	Valid values	Description
`use_wide_wire_to_pad_pin`	`same_as_macro_pin | true | false`	Controls pin tapering to pad pins. The default is `same_as_macro_pin`.
`use_wide_wire_to_port`	`same_as_macro_pin | true | false`	Controls pin tapering to ports. The default is `same_as_macro_pin`.
Width and spacing options		
`diagonal_min_width`	`true | false`	Specifies whether to use diagonal distance for minimum width violations. The default is `true`.
`ignore_drc`	`{{same_net_metal_space | same_net_enclosed_cut_space | all_same_net_drc_for_frozen_net true | false}...}`	Controls whether the router ignores specific design rule constraints (DRCs). By default (false), none of the DRCs are ignored.
`ignore_var_spacing_to_blockage`	`true | false`	Specifies whether variable route rule spacing is ignored against blockages. The default is `false`.
`ignore_var_spacing_to_pg`	`true | false`	Specifies whether variable route rule spacing is ignored against power and ground nets. The default is `false`.

Table 2-9 Detail Route Options (Continued)

Option	**Valid values**	**Description**
`use_default_width_for_ min_area_min_len_stub`	`true | false`	Specifies whether to use default width stubs to fix minimum area and minimum length violations. The default is `false`.
`var_spacing_to_same_net`	`true | false`	Specifies if variable route rule spacing is to be applied to shapes of the same net. The default is `false`.

For more information about the `set_route_zrt_detail_options` command, see the man page.

Setting Signal Integrity Options

By default, Zroute does not perform crosstalk reduction. If you enable signal integrity mode, Zroute performs crosstalk reduction during global routing and track assignment. To enable signal integrity mode, enter the following command:

```
icc_shell> set_si_options -route_xtalk_preventation true
```

When you run this command, IC Compiler automatically sets the Zroute global route and track assignment `crosstalk_driven` options to true.

The default crosstalk prevention threshold is 0.35 volts, which is probably too relaxed. If your design has crosstalk violations, you should use the `set_si_options -route_xtalk_prevention_threshold` command to lower the crosstalk prevention threshold during track assignment to a value between in the range of 0.25 to 0.35 volts. When you set the crosstalk prevention threshold, IC Compiler automatically sets the Zroute common `threshold_noise_ratio` option.

When you enable crosstalk prevention, Zroute avoids putting long, parallel wires on adjacent tracks during track assignment. To minimize noise, track assignment estimates the potential noise with a simplified crosstalk checker and reassigns wires to reduce the potential noise using the noise threshold.

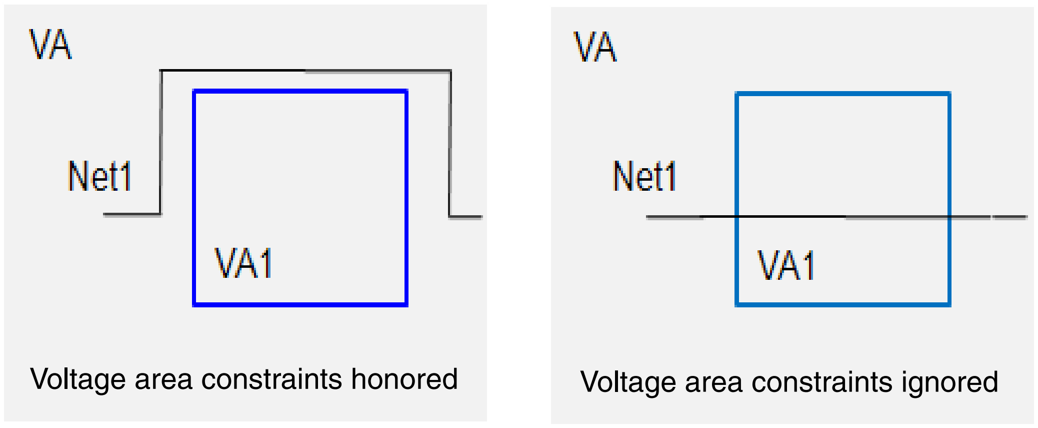
Setting Multivoltage Options

A voltage area is a placement area for one or more logical partitions that operate at the same voltage. The cells of the logical partition are associated with the partition's voltage area and are constrained to placement within that area.

By default, Zroute considers voltage area constraints during global routing, track assignment, and detail routing; however, it can violate the voltage area constraints to resolve congestion problems or DRC violations.

Figure 2-5 shows an example of routing a net in a multivoltage design. In the figure on the left, Zroute honors the voltage area constraints and routes around the other voltage area. In the figure on the right, Zroute ignores the voltage area constraints and routes through the other voltage area.

Figure 2-5 Routing a Net in a Multivoltage Design


VA
Net1
VA1
Voltage area constraints honored
VA
Net1
VA1
Voltage area constraints ignored

You can control the level of enforcement of voltage area constraints by setting the `enforce_voltage_areas` common Zroute option. This option supports the following values:

- `relaxed` (the default)

  Global routing, track assignment, and detail routing can violate the voltage area constraints to resolve congestion or DRC issues.

- `strict`

  Global routing, track assignment, and detail routing strictly follow the voltage area constraints.

- `off`

  Zroute ignores the voltage area constraints.

For example, to enable strict enforcement of the voltage area constraints, enter the following command:

```
icc_shell> set_route_zrt_common_options -enforce_voltage_areas strict
```

When you save the design in Milkyway format, the common Zroute option settings are saved with the design. For more information about setting Zroute common options, see "Setting Common Route Options" on page 2-14.

For critical nets, it might be necessary to ignore the voltage area constraints to achieve the best routing solution, even when voltage area constraints are enforced for the design. To ignore the voltage area constraints for specific nets, set the `ignore_voltage_areas` property on the nets by using the `set_zrt_net_properties` command. You can specify the nets either by using the `-nets` option to specify the nets on the command line or by specifying the nets in a file and using the `-from_file` option.

The following example shows the use of the `-nets` option:

```
icc_shell> set_zrt_net_properties -nets [get_nets my_net] \
-ignore_voltage_areas true
```

When you save the design in Milkyway format, the net property settings are saved with the design.

To get the value of the `ignore_voltage_areas` property on a net, use the `get_zrt_net_properties` command.

```
icc_shell> get_zrt_net_properties -net [get_nets my_net] \
-property ignore_voltage_areas
```

3

Using Zroute for Routing

This chapter describes how to use Zroute to perform routing tasks. It contains the following sections:

Routing Clock Nets

You can use Zroute for initial routing of clock nets, redundant via insertion on clock nets, shielding of clock nets, and ECO routing of clock nets. The following sections describe how to perform these tasks.

Note:
When you plan to use Zroute for clock tree routing, you must use Arnoldi delay calculation for the clock nets. To enable Arnoldi delay calculation for the clock nets, use the following command:

```
icc_shell> set_delay_calculation -clock_arnoldi
```

Clock Net Routing

You can route clock nets before routing the rest of the nets in the design by using the `route_zrt_group -all_clock_nets` command.

```
icc_shell> route_zrt_group -all_clock_nets
```

Note:
If you have an IC Compiler-PC package, use the `route_zrt_clock_tree` command to route the clock nets; the `route_zrt_group` command is not included in the IC Compiler-PC package.

The `route_zrt_group -all_clock_nets` command runs global routing, track assignment, and detail routing on the specified nets. By default, existing global routes are ignored. To perform incremental global routing by reusing existing global routes, use the `-reuse_existing_global_route true` option. During detail routing, Zroute also performs search and repair. By default, the detail router performs a maximum of 40 iterations. Zroute stops before completing the maximum number of iterations if it determines that all violations have been fixed or that it cannot fix the remaining violations. You can control the maximum number of detail routing iterations by using the `-max_detail_route_iterations` option.

Zroute supports three modes for global routing of clock nets:

- Balanced

  To use balanced mode global routing, you must perform incremental global routing, which reuses the global route information left by the integrated clock global router during clock tree optimization.

```
icc_shell> set_delay_calculation -clock_arnoldi
icc_shell> clock_opt -only_cts -no_clock_route
icc_shell> route_zrt_group -all_clock_nets \
-reuse_existing_global_route true
```

- Comb

  To use comb mode global routing, set the `clock_topology` global route option to `comb` before routing the clock nets.

  ```
  icc_shell> set_delay_calculation -clock_arnoldi
  icc_shell> clock_opt -only_cts -no_clock_route
  icc_shell> set_route_zrt_global_options -clock_topology comb
  icc_shell> route_zrt_group -all_clock_nets
  ```

- Normal

  To use normal mode global routing, set the `clock_topology` global route option to `normal` before routing the clock nets. Note that this is the default setting for this option.

  ```
  icc_shell> set_delay_calculation -clock_arnoldi
  icc_shell> clock_opt -only_cts -no_clock_route
  icc_shell> set_route_zrt_global_options -clock_topology normal
  icc_shell> route_zrt_group -all_clock_nets
  ```

Clock Net Redundant Via Insertion

Zroute can insert redundant vias on clock nets either during or after routing. This section describes how to insert redundant vias during clock routing. For information about postroute redundant via insertion, see “Inserting Redundant Vias” on page 4-4.

To insert redundant vias on clock nets during clock routing,

1. Define the redundant vias in a nondefault routing rule by using the `-via_cuts` option of the `define_routing_rule` command.

   ```
   icc_shell> define_routing_rule clock_via_rule \
   -via_cuts {V12 "1x2" V23 "1x2" V34 "1x2" V45 "1x2" V56 "1x2"}
   ```

   Note:
   You can define only one contact code per layer. If you require multiple contact codes to access clock pins, you must use postroute redundant via insertion.

   For information about nondefault routing rules, see “Using Nondefault Routing Rules” on page 2-9.

2. Assign the nondefault routing rule to the clock nets by using the `-routing_rule` option of the `set_clock_tree_options` command.

   ```
   icc_shell> set_clock_tree_options -routing_rule clock_via_rule
   ```

3. Run clock tree synthesis.

   ```
   icc_shell> set_delay_calculation -clock_arnoldi
   icc_shell> clock_opt -only_cts -no_clock_route
   ```

4. Route the clock nets.

   ```
   icc_shell> route_zrt_group -all_clock_nets \
   -reuse_existing_global_route true
   ```

   Zroute reserves space for the redundant vias during global routing and inserts the redundant vias during detail routing.

Clock Net Shielding

Zroute uses nondefault routing rules to define the shielding width and spacing. For information about nondefault routing rules, see “Using Nondefault Routing Rules” on page 2-9.

Note:
If you do not define the shielding spacing for clock nets, Zroute adds default spacing as shield spacing on clock nets.

After defining the nondefault routing rules and routing the clock nets, use the `create_zrt_shield` command to shield the clock nets. For information about shielding nets, see “Shielding Nets” on page 4-16.

Postroute Clock Tree Optimization

When Zroute is enabled and you run the `optimize_clock_tree` command on a routed design, the `optimize_clock_tree` command uses Zroute to perform ECO routing.

If only the clock nets are routed, run the following command to perform postroute clock tree optimization and ECO routing of the clock nets:

```
icc_shell> optimize_clock_tree -routed_clock_stage detail \
-buffer_sizng -gate_sizing
```

If both the clock and signal nets are routed, run the following commands to perform postroute clock tree optimization and ECO routing of the clock and signal nets:

```
icc_shell> optimize_clock_tree \
-routed_clock_stage detail_with_signal_routes \
-buffer_sizng -gate_sizing
```

Routing Critical Nets

You can route a group of critical nets before routing the rest of the nets in the design by using the `route_zrt_group` command (or by choosing Route > Net Group Route in the GUI). You can specify the nets to route either by using the `-nets` option to specify the nets on the command line or by specifying the nets in a file and using the `-from_file` option:

```
icc_shell> route_zrt_group -nets collection_of_critical_nets
```

or

```
icc_shell> route_zrt_group -from_file file_name
```

By default, the `route_zrt_group` command ignores existing global routes and sequentially runs global routing, track assignment, and detail routing on the specified nets. To perform incremental global routing, use the `-reuse_existing_global_route true` option. To stop after global routing, use the `-stop_after_global_route true` option. During detail routing, Zroute also performs search and repair. By default, the detail router performs a maximum of 40 iterations. If Zroute determines before then that all violations have been fixed or that it cannot fix the remaining violations, it stops. You can control the maximum number of detail routing iterations by using the `-max_detail_route_iterations` option.

Routing Signal Nets

Before you route the signal nets, all clock nets must be routed without violations.

You can route the signal nets by using one of the following methods:

- Use the basic commands to perform the stand-alone routing tasks.

  To perform global routing, use the `route_zrt_global` command. To perform track assignment, use the `route_zrt_track` command. To perform detail routing, use the `route_zrt_detail` command.

  When you run a stand-alone routing command, such as `route_zrt_global` or `route_zrt_detail`, Zroute reads in the design database at the beginning of each routing command and updates the database at the end of each command. The router does not check the input data. For example, if the track assignment step is skipped and you run detail routing directly, Zroute might generate bad routing results.

  If you need to customize your routing flow or you need to run a large design step-by-step, you might want to use the stand-alone routing commands instead of `route_opt` or automatic routing.

- Use automatic routing (the `route_zrt_auto` basic command).

  The `route_zrt_auto` basic command performs global routing, track assignment, and detail routing.

  When you run `route_zrt_auto`, Zroute reads the design database before starting routing and updates the database when all routing steps are done. If you stop auto-routing before it performs detail routing, Zroute checks the input data when you restart routing with this command.

  Use the `route_zrt_auto` command when you run routing to verify convergence, congestion, and design rule quality-of-results (QoR). You also might want to use `route_zrt_auto` if congestion QoR is your main goal, rather than timing QoR.

- Use the `route_opt` core command.

  The `route_opt` command performs global routing, track assignment, detail routing, and postroute optimization.

  Use the `route_opt` command when you are finished with the feasibility runs and want to finalize the routing and perform postroute optimization.

Zroute can insert redundant vias during signal routing. For information about this capability, see "Inserting Redundant Vias" on page 4-4.

Routing Signal Nets by Using Individual Routing Commands

After you have specified your routing options, you can debug your design or monitor each routing step by performing each routing step individually. Individual routing steps are explained in the following sections:

- Global Routing
- Track Assignment
- Detail Routing

Global Routing

Before you run global routing, you must define the Zroute common options, as described in "Setting Common Route Options" on page 2-14, and the Zroute global route options, as described in "Setting Global Route Options" on page 2-19.

To perform stand-alone global routing, use the `route_zrt_global` command.

The global router divides a design into global routing cells. By default, the width of a global routing cell is the same as the height of a standard cell and is aligned with the standard cell rows.

For each global routing cell, the routing capacity is calculated according to the blockages, pins, and routing tracks inside the cell. Although the nets are not assigned to the actual wire tracks during global routing, the number of nets assigned to each global routing cell is noted. The tool calculates the demand for wire tracks in each global routing cell and reports the overflows, which are the number of wire tracks that are still needed after the tool assigns nets to the available wire tracks in a global routing cell.

By default, the `route_zrt_global` command is not timing-driven or crosstalk-driven.

- To enable timing-driven global routing, set the `timing_driven` global route option.
- To enable crosstalk-driven global routing, set the `route_xtalk_prevention` signal integrity option, as described in "Setting Signal Integrity Options" on page 2-27.

By default, existing global routes are ignored. To perform incremental global routing by reusing the existing global routes, use the `-reuse_existing_global_route true` option when you run global routing.

Global routing is done in two phases:

- The initial routing phase, in which the tool routes the unconnected nets and calculates the overflow for each global routing cell
- The rerouting phase, in which the tool tries to reduce congestion by ripping up and rerouting nets around global routing cells with overflows

  The tool might perform several rerouting iterations. At the end of each rerouting iteration, the tool recalculates the overflows. You should see a reduction in the total number of global routing cells with overflow and in the total overflow numbers. The global router stops and exits from the reroute phase when the congestion is solved (or cannot be solved further) or after the maximum number of iterations, as defined by the `-effort` option. By default, the tool performs a maximum of four reroute iterations (medium effort). You can perform up to five reroute iterations by specifying high effort.

  There are four global routing effort levels: minimum, low, medium, and high.

  - Minimum (`-effort minimum`)

    The minimum effort level uses two times larger global routing cells relative to the other effort levels. It also has a much lower congestion cost and runs only one reroute iteration. It should only be used for an initial congestion evaluation, not for detail routing.

  - Low (`-effort low`)

    Low effort runs only two reroute iterations with very similar congestion cost. It is faster in comparison to medium effort and has reasonable QoR. If your design is not very congested, you can use the low effort level.

- Medium (`-effort medium`)

  Medium effort is the default effort level and runs a maximum of three reroute iterations. Global routing stops after the third iteration or when the overflow is resolved, whichever occurs first.

- High (`-effort high`)

  High effort runs up to four reroute iterations. If your design is very congested, use the high effort level.

The tool reports design statistics and congestion data after completing each iteration.

The global router reports design statistics and congestion data after the initial routing phase and after each reroute iteration. When global routing is completed, the global router reports a summary of the wire length and via count. Example 3-1 shows a sample global routing report.

Example 3-1 Global Routing Report

```
Start Global Route ...
[Init] Elapsed real time: 0:00:00
[Init] Elapsed cpu  time: sys=0:00:00 usr=0:00:00 total=0:00:00
......
Begin global routing.
Constructing data structure ...
Design statistics:
Design Bounding Box (0.00,0.00,3180.00,1154.00)
Number of routing layers = 10
layer M1, dir Ver, min width = 0.09, min space = 0.09 pitch = 0.24
layer M2, dir Hor, min width = 0.10, min space = 0.10 pitch = 0.20
layer M3, dir Ver, min width = 0.10, min space = 0.10 pitch = 0.30
layer M4, dir Hor, min width = 0.10, min space = 0.10 pitch = 0.30
layer M5, dir Ver, min width = 0.10, min space = 0.10 pitch = 0.30
layer M6, dir Hor, min width = 0.10, min space = 0.10 pitch = 0.30
layer M7, dir Ver, min width = 0.10, min space = 0.10 pitch = 0.30
...
Net statistics:
Total number of nets     = 231242
Number of nets to route  = 231177
Number of single or zero port nets = 64
1 nets are partially routed.
1 nets are partially drouted.
0 nets are partially grouted.
1 nets are routed.
1 nets are drouted.
0 nets are grouted.
4 nets have non-default rule non_m2_route_rules_for_noise
12 nets have non-default rule clockRouteRuleCTS
...
```

```
phase3. Routing result:
phase3. Both Dirs: Overflow =   453 Max = 4 GRCs =   449 (0.02%)
phase3. H routing: Overflow =   148 Max = 1 (GRCs = 114) GRCs =   202
(0.02%)
phase3. V routing: Overflow =   304 Max = 4 (GRCs =  2) GRCs =   247
(0.03%)
phase3. M1         Overflow =     1 Max = 0 (GRCs =  2) GRCs =     2
(0.00%)
phase3. M2         Overflow =    34 Max = 0 (GRCs = 88) GRCs =    88
(0.01%)
phase3. M3         Overflow =   184 Max = 4 (GRCs =  2) GRCs =   126
(0.01%)
phase3. M4         Overflow =    80 Max = 1 (GRCs = 80) GRCs =    80
(0.01%)
phase3. M5         Overflow =     0 Max = 0 (GRCs =  0) GRCs =     0
(0.00%)
phase3. M6         Overflow =    34 Max = 1 (GRCs = 34) GRCs =    34
(0.00%)
phase3. M7         Overflow =   119 Max = 1 (GRCs = 119) GRCs =   119
(0.01%)
......phase3. Total Wire Length = 16820202.00
phase3. Layer M1 wire length = 10412.66
phase3. Layer M2 wire length = 3576314.50
phase3. Layer M3 wire length = 2721198.75
phase3. Layer M4 wire length = 2988112.75
phase3. Layer M5 wire length = 2201333.00
phase3. Layer M6 wire length = 4374871.00
phase3. Layer M7 wire length = 947959.25
phase3. Layer M8 wire length = 0.00
phase3. Layer M9 wire length = 0.00
phase3. Layer M10 wire length = 0.00
phase3. Total Number of Contacts = 1820147
phase3. Via VIA12 count = 749084
phase3. Via VIA23 count = 762968
phase3. Via VIA34 count = 142823
phase3. Via VIA45 count = 91356
phase3. Via VIA56 count = 46191
phase3. Via VIA67 count = 27725
phase3. Via VIA78 count = 0
phase3. Via VIA89 count = 0
phase3. Via VIA910mH count = 0
phase3. completed.
```

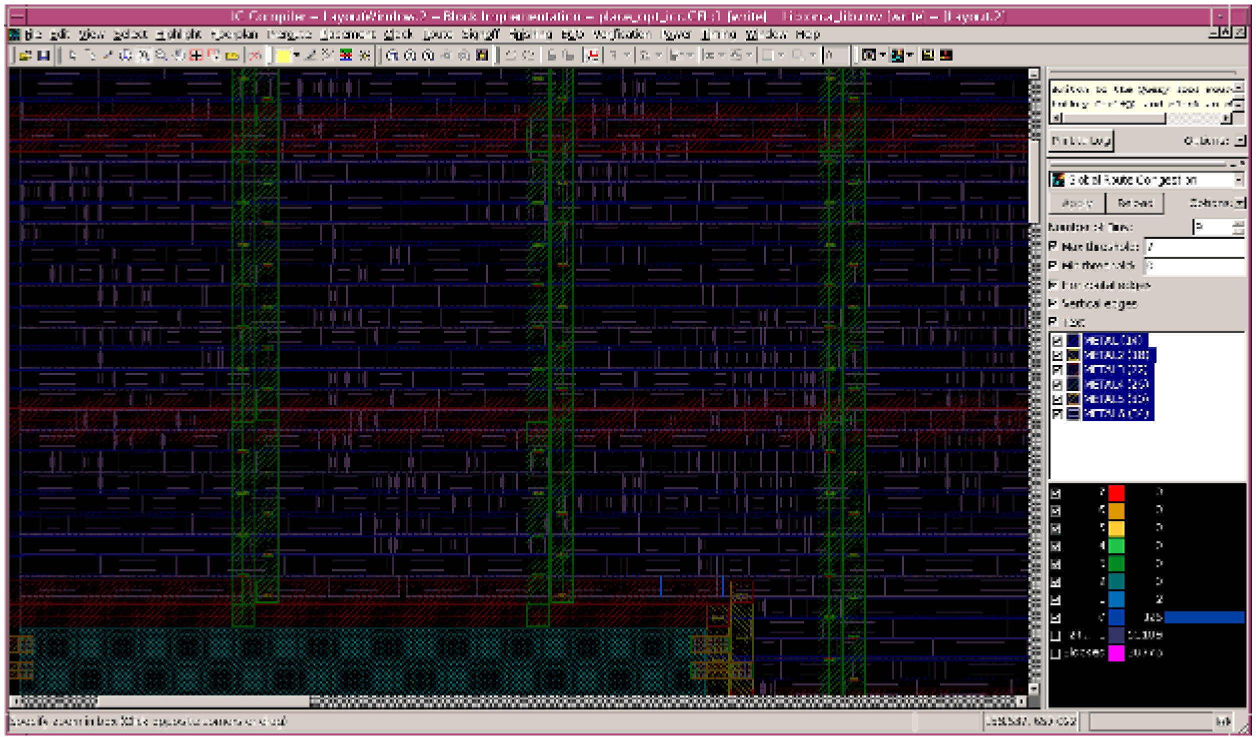
At the end of global routing, Zroute generates g-links and g-vias on each routed net for the next routing step.

After Zroute finishes global routing, it generates a layer-based congestion map in which you can simultaneously see the overflow distribution for one or more layers. Before proceeding to detail routing, display the congestion map in the GUI and check the overflow distribution. The congestion report and map will help you to identify congested areas.

To display the layer-based congestion map, choose Route > Global Route Congestion Map in the GUI. Figure 3-1 shows a sample congestion map.

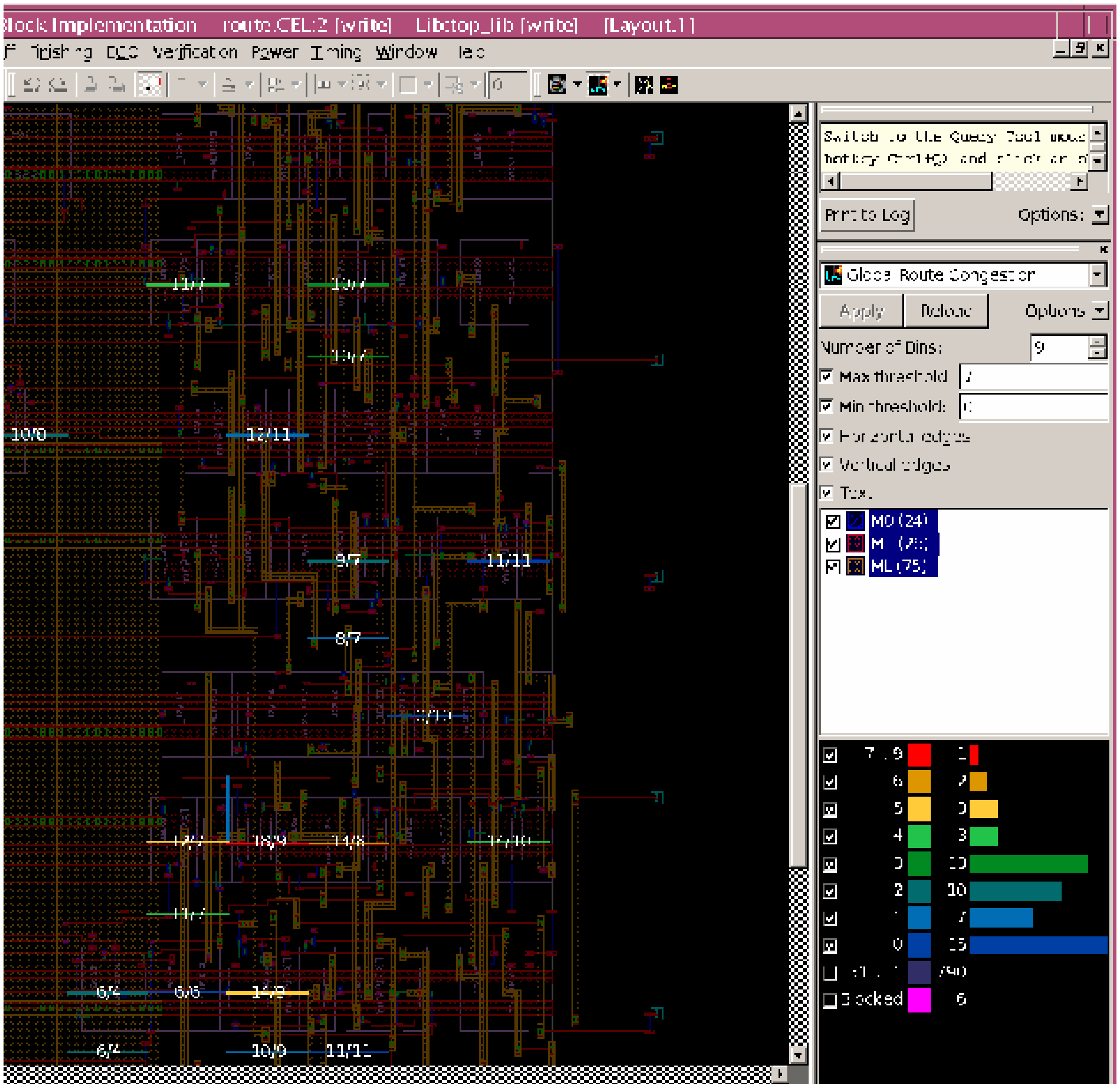
By default, the congestion map is displayed for all metal layers and all congestion buckets. You can customize the congestion map by selecting or deselecting the metal layers and congestion buckets in the Map Mode dialog box.

Figure 3-1 Global Route Congestion Map



If the design shows congested areas, zoom into the congested area to see the overflow number on the global routing cell. For example, in Figure 3-2, the red highlight on the edge of the global routing cell shows 18/9. This means there are 9 wire tracks available, but 18 tracks are needed.

Figure 3-2 Global Route Overflow on Global Routing Cell



Switch to the Query Tool mode hotkey (Ctrl+Q) and click on a...

Number of Bins: 9
☑ Max threshold: 7
☑ Min threshold: 0
☑ Horizontal edges
☑ Vertical edges
☑ Text

☑	7 - 9	🟥	1
☑	6	🟧	2
☑	5	🟨	3
☑	4	🟩	3
☑	3	🟩	33
☑	2	🟦	10
☑	1	🟦	7
☑	0	🟦	35
☐	[illegible]	⬛	790
☐	Blocked	🟪	6

Track Assignment

Before you run track assignment, you must define the Zroute common options, as described in "Setting Common Route Options" on page 2-14, and the Zroute track assignment options, as described in "Setting Track Assignment Options" on page 2-21.

To perform standalone track assignment, run the `route_zrt_track` command (or choose Route > Track Assignment in the GUI).

The main task of track assignment is to assign routing tracks for each global routed net. During track assignment, Zroute performs the following tasks:

- Assigns tracks in horizontal partitions.
- Assigns tracks in vertical partitions.
- Reroutes overlapping wires.

After track assignment finishes, all nets are routed but not very carefully. There are many violations, particularly where the routing connects to pins. Detail routing works to correct those violations.

By default, the `route_zrt_track` command is not timing-driven or crosstalk-driven.

- To enable timing-driven mode, set the `timing_driven` track assignment option.
- To enable crosstalk-driven mode, set the `route_xtalk_prevention` signal integrity option, as described in "Setting Signal Integrity Options" on page 2-27.

At the end of track assignment, Zroute reports a summary of the wire length and via count. Example 3-2 shows a sample track assignment report.

Example 3-2 Track Assignment Report

```
Wire length and via report:
---------------------------
Number of metal1 wires: 90644           via01: 0
Number of metal2 wires: 1305119         via12: 1171053
Number of metal3 wires: 823570          via23: 1300628
Number of metal4 wires: 124268          via34: 192435
Number of metal5 wires: 28001           via45: 44130
Number of metal6 wires: 4054            via56: 7644
Number of metal7 wires: 0               via67: 0
Total number of wires: 2375656           vias: 2715890
Total metal1 wire length: 170860.9
Total metal2 wire length: 2493004.5
Total metal3 wire length: 4127489.2
Total metal4 wire length: 2109298.0
Total metal5 wire length: 1584743.5
Total metal6 wire length: 300718.1
Total metal7 wire length: 0.0
Total wire length: 10786114.0

Longest metal1 wire length: 540.7
Longest metal2 wire length: 603.5
Longest metal3 wire length: 729.7
Longest metal4 wire length: 821.1
Longest metal5 wire length: 958.2
Longest metal6 wire length: 935.7
Longest metal7 wire length: 0.0
```

Detail Routing

Before you run detail routing, you must define the Zroute common options, as described in "Setting Common Route Options" on page 2-14, and the Zroute detail route options, as described in "Setting Detail Route Options" on page 2-21.

The detail router uses the general pathways suggested by global routing and track assignment to route the nets, and then it divides the design into partitions and looks for DRC violations in each partition. When the detail router finds a violation, it rips up the wire and reroutes it to fix the violation. During detail routing, Zroute concurrently addresses routing design rules and antenna rules and optimizes via count and wire length. For more information about antenna rules and via count and wire length optimization, see Chapter 4, "Using Zroute for Design for Manufacturing and Chip Finishing."

To perform stand-alone detail routing, run the `route_zrt_detail` command (or choose Route > Detail Route in the GUI).

By default, detail routing is performed on the whole design. You can restrict the routing to a specific area of the design by using the `-coordinates` option (or by specifying or selecting the bounding box in the GUI).

By default, Zroute uses one uniform partition for the first iteration, which is needed to generate all DRC violations for the chip at the same time. At the beginning of each subsequent iteration, the router checks the distribution of the DRC violations. If the DRC violations are evenly distributed, the detail router uses a uniform partition. If the DRC violations are located in some local areas, the detail router uses nonuniform partitions. The detail router repeats this process until one of the following conditions exists:

- All of the violations have been fixed
- It cannot fix any of the remaining violations
- The maximum number of iterations has been reached

You can run additional detail routing iterations by running incremental detail routing (the `-incremental` option). Be sure to use the `-incremental` option; otherwise, Zroute restarts at iteration 0 with a fixed-size partition.

```
icc_shell> route_zrt_detail -incremental true
```

Note:
Incremental detail routing does not fix open nets. To fix open nets, you must run ECO routing. For information about ECO routing, see "Performing ECO Routing" on page 3-22.

By default, the maximum number of detail routing iterations is 40. You can change this limit by setting the `-max_number_iterations` option. You can force the detail router to complete the specified iterations, regardless of the DRC convergence status, by setting the `force_max_number_iterations` detail route option to true.

```
icc_shell> route_zrt_detail -max_number_iterations 20
icc_shell> set_route_zrt_detail_options -force_max_number_iterations true
```

By default, the `route_zrt_detail` command is not timing-driven. To enable timing-driven detail routing, set the `timing_driven` detail route option.

```
icc_shell> set_route_zrt_detail_options -timing_driven true
```

If you want to view the DRC violations before postroute optimization, you can save the design after a specified number of iterations by setting the `save_after_iterations` detail route option. The saved design is called DR_itr*n*, where n is the specified iteration. You can use a string other than DR as the prefix by setting the `save_cell_prefix` detail route option.

Zroute generates a DRC violations summary at the end of each iteration. For more information about the DRC violations reported by Zroute, see "Verifying the Routing" on page 3-24. Before reporting the final DRC violations, Zroute merges redundant violations. Example 3-3 shows a sample detail routing report.

Example 3-3 Detail Routing Report

```
Start DR iteration 0: uniform partition
Routed  1/5734 SBoxes, Violations =     0
Routed  28/5734 SBoxes, Violations =    14
… …
DRC-SUMMARY:
        @@@@@@ TOTAL VIOLATIONS =      437
        @@@ Total number of instance ports with antenna violations = 690
          Diff net spacing : 300
                 End of line spacing : 1
                 Same net spacing : 6
                 Same net via-cut spacing : 2
                 Less than minimum area : 20
                 Less than minimum edge length : 19
                 Needs fat contact : 5
                 Needs fat contact on extension : 7
                 Internal-only types : 77

                 End DR iteration 21 with 198 parts
        @@@@ Total nets not meeting constraints =       1
Stop DR since not converging
           … …
                 Information: Merged away 13 aligned/redundant DRCs.
(RT-805)
     DR finished with 47 violations and 144 instance ports antenna
violations

          Diff net spacing : 31
          Less than minimum edge length : 1
          Internal-only types : 15

Total Wire Length =                    11107954 micron
Total Number of Contacts =             2787466
Total Number of Wires =                2759250
Total Number of PtConns =              598905
Total Number of Non-Default Contacts = 362
                Layer           M1 :     190733 micron
                Layer           M2 :    2474471 micron
                Layer           M3 :    4106791 micron
                Layer           M4 :    2377087 micron
                Layer           M5 :    1701419 micron
                Layer           M6 :     257452 micron
                Layer           M7 :          0 micron
                Via          VIA56 :       9778
                Via          VIA45 :      76055
```

Routing Signal Nets by Using Automatic Routing

Before you use automatic routing to route the signal nets, you must define the Zroute options (common, global route, track assignment, and detail route), as described in "Setting Zroute Options" on page 2-14.

To run automatic routing, use the `route_zrt_auto` command (or choose Route > Auto Route in the GUI). By default, the `route_zrt_auto` command ignores existing global routes and sequentially invokes global routing, track assignment, and detail routing. You can perform incremental global routing by using the `-reuse_existing_global_route true` option. You can stop after track assignment by using the `-stop_after_track_assignment` option.

By default, the `route_zrt_auto` command is not timing-driven or crosstalk-driven.

- To enable timing-driven mode, set the `timing_driven` global route, track assignment, and detail route options.
- To enable crosstalk-driven mode for global routing and track assignment, set the `route_xtalk_prevention` signal integrity option, as described in "Setting Signal Integrity Options" on page 2-27.

Table 3-1 describes the `route_zrt_auto` options. For more information, see the man page.

Table 3-1 route_zrt_auto Command Options

Command option	GUI object	Description
`-reuse_existing_global_route true | false`	check box	Performs incremental global routing. The default is `false`.
`-stop_after_track_assignment true | false`	"Stop after track assignment" check box	Stops after performing track assignment. The default is `false`.
`-save_after_global_route true | false`	"Save cell after: Global route" check box	Controls whether the design is saved after global routing. When true, the tool saves the design in a CEL view named auto_GR. The default is false.
`-save_after_track_assignment true | false`	"Save cell after: Track assignment" check box	Controls whether the design is saved after track assignment. When true, the tool saves the design in a CEL view named auto_TA. The default is false.

Table 3-1 route_zrt_auto Command Options (Continued)

Command option	**GUI object**	**Description**
`-save_after_detail_route true | false`	"Save cell after: Detail route" check box	Controls whether the design is saved after detail routing. When true, the tool saves the design in a CEL view named auto_DR. The default is false.
`-max_detail_route_iterations` *`cnt`*	"Maximum detail router iterations" box	The maximum number of detail routing iterations. The default is the 40.
`-save_cell_prefix` *`string`*	"Save cell prefix" box	Specifies the prefix to use for intermediate saved designs. The default is *auto*.

Routing Signal Nets by Using route_opt

Before you use the `route_opt` command to route the signal nets, you must define the Zroute options (common, global route, track assignment, and detail route), as described in "Setting Zroute Options" on page 2-14 and set the `route_opt` routing and optimization strategy, as described in the following section.

Setting the Routing and Optimization Strategy

IC Compiler provides strategy controls that you set to guide how routing and postroute optimizations are run.

To set the routing and optimization strategy, use the `set_route_opt_strategy` command (or choose Route > Set Core Routing and Optimization Strategy in the GUI).

Table 3-2 describes the `set_route_opt_strategy` options. For more information, see the man page.

Table 3-2 set_route_opt_strategy Options

Command option	GUI object	Description
`-search_repair_loops`	"Maximum initial route Search and Repair cycles" box	The maximum number of initial routing iterations. The Zroute default is 10 (the classic router default is 15).
`-eco_route_search_ repair_loops`	"Maximum ECO Search and Repair cycles" box	The maximum number of ECO iterations. The Zroute default is 4 (the classic router default is 5).
`-fix_hold_mode all | route_base`	"Fix hold optimization stage" area	The stages for which hold optimization is performed. You can select prerouting, global routing, and detail routing (`all`) or global routing and detail routing (`route_base`). The default is `route_base`.
`-xtalk_reduction_loops`	"Maximum crosstalk reduction cycles" box	The maximum number of crosstalk reduction optimization cycles. The default is 2.
`-route_drc_threshold`	"Route violations threshold to trigger the reduction Search and Repair loop" box	The threshold number of routing violations before limiting detail routing to one iteration. The default is 3000.

Note:
Because Zroute performs wire and via optimization concurrently with detail routing rather than in a separate step, the following options of the `set_route_opt_strategy` command do not apply to Zroute: `-optimize_wire_via_search_repair_loops` and `-route_run_time_limit`.

To report your settings, use the `report_route_opt_strategy` command.

Running route_opt

To run routing and postroute optimization, use the `route_opt` command (or choose Route > Core Routing and Optimization in the GUI).

By default, the `route_opt` command runs global routing, track assignment, and detail routing in timing-driven mode, followed by postroute optimization. To disable timing-driven mode, you must set the `timing_driven` Zroute option to false. There are separate `timing_driven` options for global routing, track assignment, and detail routing.

Note:
By default, the Zroute basic routing commands are not timing-driven.

If you want to analyze the routing results before performing postroute optimization, first run `route_opt` without postroute optimization (`-initial_route_only` option), and then run `route_opt` with postroute optimization only (`-skip_initial_routing` option).

```
icc_shell> route_opt -initial_route_only
# analyze routing results
icc_shell> route_opt -skip_initial_routing
```

Table 3-3 describes the `route_opt` options. For more information, see the man page.

Table 3-3 route_opt Command Options

Command option	GUI object	Description
`-effort low | medium | high`	Effort radio buttons	Specifies the optimization effort level. Higher effort levels provide more aggressive optimization at a runtime cost. The default is `medium`.
`-stage global |track | detail`	"Run optimization after" radio buttons	Specifies the last routing stage performed by `route_opt` before performing optimization.
`-xtalk_reduction`	"Crosstalk reduction and SI optimization" radio button	Performs crosstalk reduction optimization in addition to signal integrity optimization. By default, `route_opt` does not perform signal integrity optimization.
`-only_xtalk_reduction`	"Crosstalk reduction optimization only" radio button	Performs only crosstalk reduction optimization (not signal integrity optimization).
`-power`	"Perform power optimization" check box	Performs leakage power optimization.

Table 3-3 route_opt Command Options (Continued)

Command option	GUI object	Description
`-skip_initial_route`	"Skip initial routing" check box	Performs postroute optimization without routing.
`-initial_route_only`	"Initial routing only" check box	Performs only initial routing without postroute optimization. This option works in conjunction with the `-stage` option. A maximum of 10 detail routing iterations are run.
`-size_only`	"Sizing only optimization" check box	Resolves setup time violations by sizing only.
`-optimize_wire_via`	"Optimize wire and via routing" check box	This option is not supported by Zroute. Wire and via optimization is done concurrently with detail routing.
`-area_recovery`	"Recover area for cells that are not on critical timing paths" check box	Recovers area for cells not on critical paths. To restrict area recovery to high-density areas, set the `physopt_density_area_recovery` variable to true. You can also use this variable with `-only_area_recovery`.
`-wire_size`	"Use wire sizing to fix setup time violations" check box	Determines whether wire sizing is used to fix setup time violations.
`-incremental`	"Incremental mode" check box	Performs incremental postroute optimization.
`-incremental -only_wire_size`	"Timing optimization only with wire size" radio button	Performs only wire sizing to resolve setup time violations during incremental mode.
`-incremental -only_hold_time`	"Hold timing optimization only" radio button	Resolves only hold time violations during incremental mode. Use the `set_fix_hold` command to enable hold time fixing.
`-incremental -only_area_recovery`	"Area recovery only" radio button	Performs only area recovery during incremental mode.

Table 3-3 route_opt Command Options (Continued)

Command option	GUI object	Description
`-incremental -only_design_rule`	"Fix design rules only" radio button	Performs only logical design rule fixing during incremental mode.
`-incremental -only_power_recovery`	"Power recovery only" radio button	Performs only power recovery during incremental mode.
`-save_after_iterations`	N/A	Specifies the iteration after which the design is saved. The default is 1 (the second iteration).
`-save_cell_prefix`	N/A	Specifies the prefix for the saved design. The saved design is named *prefix*_INIT_RT. The default prefix is the cell name.
`-num_cpus` *number*	"Number of CPUs" box	This option is not supported by Zroute. Zroute uses multithreading rather than distributed routing. For information about enabling multithreading, see "Enabling Multithreading" on page 2-3.

Saving the Intermediate Results

By default, when you run `route_opt` with Zroute, the tool automatically saves the design after two iterations (iteration 1) to allow you to view the DRC violations before postroute optimization. The saved design is called *cell_name*_INIT_RT_itr1. You can select an iteration other than iteration 1 by using the `-save_after_iterations` option. You can use a string other than the cell name as the prefix by using the `-save_cell_prefix` option. These options are supported only when using `route_opt` with Zroute.

Note:
If you run `route_opt -intial_route_only`, the tool does not automatically save the design after two iterations. However, you can force the tool to save the design by using the `-save_after_iterations` option.

In addition, Zroute supports the standard `route_opt` checkpointing. To enable `route_opt` checkpointing, set the `routeopt_checkpoint` variable to true.

If you also use the `-save_after_iterations` and `-save_cell_prefix` options, they are honored only for the first version of the design saved during checkpointing. They are not honored for subsequent checkpoint saves.

Performing ECO Routing

Whenever you modify the nets in your design, you need to run engineering change order (ECO) routing to reconnect the routing.

To run ECO routing, use the `route_zrt_eco` command (or choose Route > ECO Route in the GUI). Zroute can reserve space for redundant vias during ECO routing. For information about this capability, see "Concurrent Soft-Rule-Based Redundant Via Insertion" on page 4-7.

By default, the `route_zrt_eco` command connects open nets and then fixes design rule violations in the entire design. To limit DRC fixing to the neighborhood of the open nets, set the `-open_net_driven` option to `true`.

The `route_zrt_eco` command routes the newly added wires by running them sequentially through global routing, track assignment, and detail routing. By default, the `route_zrt_eco` command ignores existing global routes. To perform incremental global routing, use the `-reuse_existing_global_route true` option. If a net has an ECO change, the global router does not reroute detail routed wires to relieve congestion; instead, it reuses the dangling wires of the same net. To disable the reuse of dangling wires, set the `-utilize_dangling_wires` option to false.

By default, `route_zrt_eco` reroutes any wires, whether modified or not, to fix DRC violations. You can change the scope of rerouting performed by the ECO router by setting the `-reroute` option:

- To limit rerouting to modified wires, use `-reroute modified_nets_only`.
- To first attempt to fix DRC violations by rerouting modified nets and then reroute other nets if necessary, use `-reroute modified_nets_first_then_others`.

  A common use for this option is to route the clock nets affected by an ECO in a fully routed design.

Zroute generates a DRC violations summary at the end of each detail routing iteration. Before reporting the final DRC violations, Zroute merges redundant violations. For more information about the DRC violations reported by Zroute, see "Verifying the Routing" on page 3-24.

Table 3-4 describes the `route_zrt_eco` options. For more information, see the man page.

Table 3-4 route_zrt_eco Command Options

Command option	GUI object	Description
`-nets` *`nets`*	"Nets" section	The nets to verify. If you specify this option, the ECO router always fixes DRC violations for the whole design. By default, all nets in the design are verified.
`-open_net_driven true | false`	"Open net mode" radio buttons	Controls whether the ECO router fixes DRC violations for the whole design or only in the neighborhood of open nets. By default (false), ECO routing fixes DRC violations for the whole design.
`-max_detail_route_iterations`	"Maximum detail route iterations" box	The maximum number of detail routing iterations. The default is the 40.
`-reroute modified_nets_only | modified_nets_first_then_others | any_nets`	"Reroute nets" radio buttons	Controls which nets can be rerouted to fix DRC violations. The default is `any_nets`.
`-utilize_dangling_wires true | false`	"Utilize dangling wires" check box	Controls whether the router tries to reuse existing dangling routes to fix open nets. The default is true.

Analyzing the Routing Results

You can analyze the routing results by reporting on the cell placement and routing statistics or by running routing verification. The following sections describe how to perform these tasks.

Reporting Cell Placement and Routing Statistics

You can report on cell placement and routing statistics to help you determine whether to perform further optimizations to improve timing. For example, if the statistics show that an area has high congestion, an additional optimization might not have room to add or size up standard cells.

To view the place and route summary report, run the `report_design -physical` command (or choose Design > Report Design in the GUI, and then select “Show physical”).

Verifying the Routing

After you have completed routing with Zroute, you can check the design for routing design rule violations.

IC Compiler provides the following methods for verifying the routing:

- Use the Hercules tool to check the routing design rules defined in the foundry runset.

  To use this method, run the `signoff_drc` command or choose Verification > Signoff DRC in the GUI.

  Note:
  A Hercules license is required to run the `signoff_drc` command.

- Use the IC Compiler DRC checking engine to check the routing design rules defined in the Milkyway technology file.

  To use this method, run the `verify_zrt_route` command or choose Route > Verify Route in the GUI.

- Import the DRC checking results from the Calibre tool.

The following sections describe these methods.

Using the signoff_drc Command

The `signoff_drc` command performs routing DRC checking by using the Hercules tool with the foundry runset and reports all the errors in the top-level design by expanding the lower-level errors without merging them. A Hercules license is required to run the `signoff_drc` command. For more information about Hercules, see the Hercules documentation, which is available on SolvNet.

To use `signoff_drc` to perform DRC checking,

- Set up the Hercules environment.
- Set up the physical signoff options.
- Run the `signoff_drc` command.

The following sections describe these tasks.

Setting Up the Hercules Environment

The `signoff_drc` command runs the Hercules tool. In order to do this, you must have a Hercules license, and you must specify the location of the Hercules executable by setting the `HERCULES_HOME_DIR` environment variable. You can set this variable in your .cshrc file. For example,

```
setenv HERCULES_HOME_DIR /root_dir/hercules_2006.12_SP2-4
set path = ($path $HERCULES_HOME_DIR/bin/AMD.64)
```

You must ensure that the version of the Hercules executable that you specify is compatible with the version of IC Compiler that you are using.

Setting Up The Physical Signoff Options

To set up the physical signoff options, use the `set_physical_signoff_options` command (or choose Verification > Set Physical Signoff Options in the GUI).

Table 3-5 shows the physical signoff options that apply to the `signoff_drc` command.

Table 3-5 Physical Signoff Options for signoff_drc

Command option	Description
`-drc_runset` *`filename`* (DRC box in GUI “Foundry runset” section)	Specifies the foundry runset to use for DRC checking (required).
`-mapfile` *`filename`* (“Layer mapping file” box in GUI)	Specifies the layer mapping file (optional). For more information, see “Defining the Layer Mapping Between Milkyway and the Hercules Runset.”

Table 3-5 Physical Signoff Options for signoff_drc (Continued)

Command option	Description
`-dp_hosts {host_list}` (Hosts box in GUI)	Defines the distributed processing options (optional). For more information, see "Performing Distributed DRC Checking."
`-num_cpus int` ("Number of CPU's" box in GUI)	

To report the current settings for the physical signoff options, use the `report_physical_signoff_options` command.

Defining the Layer Mapping Between Milkyway and the Hercules Runset

In general, the Milkyway technology file and the Hercules runset use the same layer numbers. If they do not, you must supply a layer mapping file to map the Milkyway layers to the Hercules layers.

The `signoff_drc` command accepts a layer mapping file in either Hercules or Milkyway format.

The syntax of the mapping information in a Hercules mapping file is

```
Milkyway_layer_list > Hercules_layer_list
```

The rules for specifying the layer lists are

- Each layer in a layer list must be separated by a space. Or, if specified in a range, the range must consist of a starting layer and an ending layer joined by a dash (-).
- Generally, parentheses should surround the layer list. However, you can omit parentheses if you specify only one layer.

You can include comments in a layer mapping file by preceding the comment with a pound sign (#). All text that follows a pound sign on a given line is a comment.

The syntax of the mapping information in a Milkyway mapping file is:

```
Milkyway_layer Hercules_layer
```

Performing Distributed DRC Checking

By default, `signoff_drc` uses a single process to perform DRC checking. To reduce the runtime used for DRC checking, you can use distributed processing. To enable distributed processing, you must specify the number of CPUs to use (`-num_cpus` option) and identify the CPUs to be used (`-dp_hosts` option). The number of CPUs that you specify must be less than or equal to the number of hosts that you specify. To use multiple CPUs on a given host, you must include the host name multiple times in the host list.

Running the signoff_drc Command

By default, the `signoff_drc` command uses the FRAM view to check all routing layers of the entire chip for all rules specified in the foundry Hercules runset. To see problems that are masked by the FRAM view abstraction, use the CEL view instead of the FRAM view by using the `-read_cell_view` option (or by selecting the “Cell view” radio button in the GUI).

Note:
The `signoff_drc` command uses the on-disk design information, not the design information in memory. You must save the current state of the design before running the `signoff_drc` command.

You can restrict the DRC checking to specific areas of the chip or to specific design rules.

- To restrict the checking to specific areas of the chip,
  - If you are using the command line, use the `-bounding_box` option to specify the coordinates of each area. You can specify multiple areas by specifying the coordinates for each area.
  - If you are using the GUI, select the “Areas” check box and specify the coordinates for each area in the “Bounding Boxes” list. To specify an area, either draw the bounding box or enter the x- and y-coordinates of the box. For each area, you can also specify that the bounding box should snap to the minimum grid (the default), placement site, routing track, middle routing track, or user grid.
- To restrict the checking to specific rules,
  - To check only the specified rules, specify the rules in the `-select_rule` option (or in the Pattern box in the “To use” section of the Rules area in the GUI).
  - To check all rules except the specified rules, specify the rules in the `-unselect_rule` option (or in the Pattern box in the Excluded section of the Rules area in the GUI).

  In either case, you specify the rules by specifying a matching pattern for the rule names. The rule names are specified in the COMMENT section in the Hercules runset file.

  For example, to restrict `signoff_drc` to route validation, select the metal layer rules and exclude the metal density rules.

After you complete DRC checks on the routing layers, you can perform a quick DRC check on the Milkyway database by rerunning the `signoff_drc` command with the `-check_all_layers` option (the "Check DRC violations on all layers" check box in the GUI). When you use this option, the design information comes from the CEL view, and Hercules checks all layers, including the nonrouting layers.

The `signoff_drc` command creates a Milkyway error cell file that you can use to report or display the DRC violations. By default, the error cell file generated by `signoff_drc` is called *design*_sdrc.err. You can control this name by using the `-error_cell` option (or the "Error cell name" box in the GUI). For information about using the error cell file to debug DRC violations, see "Analyzing DRC Violations" on page 3-31.

In addition to the error cell file, the `signoff_drc` command generates the following files, which you can use for debugging, and places them in a signoff_drc_run directory under the current working directory:

- *design*.errsum or *design*.LAYOUT_ERRORS

  These files contain an error summary report.

- sdrc.ev

  This file contains the Hercules runset file, including any options added by the `signoff_drc` command.

- sdrc.log

  This file contains any errors that occurred during the `signoff_drc` run.

- ./run_details/*design*.sum

  This file contains the detailed Hercules log file.

Using the verify_zrt_route Command

The `verify_zrt_route` command runs an internal checker that checks for routing DRC violations, unconnected nets, antenna rule violations, and voltage area violations.

By default, the routing is verified for all nets in the design, except those marked as user nets. To verify the routing only for specific nets, specify the nets by using the `-nets` option (or by specifying the nets in the "Nets" box in the GUI). To check user routes, set the `-check_from_user_shapes` option to true.

Note:
The classic router uses center lines to determine connectivity, but Zroute does not. Do not run the `verify_route` command on a design routed with Zroute; otherwise you will see many DRC violations due to the different connectivity models between Zroute and the classic router.

The `verify_zrt_route` command reports the following DRC violations:

- Spacing violations
  - Different-net spacing
  - Different-net variable rule spacing
  - Different-net via-cut spacing
  - Different-net fat extension spacing
  - Dog bone spacing
  - End-of-line spacing
  - Enclosed via spacing
  - Same-net spacing
  - Same-net via-cut spacing
  - Same-net fat extension spacing
  - Special notch spacing
  - U-shape spacing
  - Via-cut to metal spacing
  - Soft spacing
- Area violations
  - Less than minimum area
  - Less than minimum enclosed area
  - Fat wire via keepout area
  - Jog wire via keepout area
- Length and width violations
  - Less than minimum width
  - Less than minimum length
  - Less than minimum edge length
  - Protrusion length

- Contact violations
  - Needs fat contact
  - Needs poly contact
  - Needs fat contact on extension
  - Over maximum stack level
- Enclosure violations
  - End-of-line wire via enclosure
  - Jog wire via enclosure
  - T-shape wire via enclosure
- Others
  - Open nets
  - Antenna violations
  - Nets crossing the top-cell boundary
  - Frozen layers
  - Minimum layer
  - Maximum layer
  - Voltage area violations

Note:
These violations are also reported after each detail route iteration.

By default, `verify_zrt_route` reports a maximum of 200 open nets. To report all open nets, use the `-report_all_open_nets true` option.

After you run the `verify_zrt_route` command, you can use the `report_error_coordinates` command to report the location of each violation. You can also use the error browser to examine the violations in the GUI. For information about analyzing the DRC violations, see “Analyzing DRC Violations” on page 3-31.

After running `verify_zrt_route`, you can use the following command to run incremental detail routing that uses the `verify_zrt_route` results as input:

```
icc_shell> route_zrt_detail -incremental true -initial_drc_from_input
true
```

Note:
Incremental detail routing does not fix open nets. To fix open nets, you must run ECO routing. For information about ECO routing, see the next section, “Performing ECO Routing.”

Using the Calibre Interface

You can perform DRC checking with the Calibre tool, convert the Calibre DRC error file to a Milkyway error cell file, and then report or view the errors in IC Compiler. To convert the Calibre DRC error file to a Milkyway error cell file, use the `read_drc_error_file` command (or choose Verification > Read Third-party DRC Error File in the GUI).

For example,

```
icc_shell> read_drc_error_file Calibre_error_file
```

By default, the command generates a Milkyway error cell file called *cell_name*.err, where *cell_name* is derived from the Calibre error report. You can specify a name for the Milkyway error cell file by using the `-error_cell` option.

Note:
The `read_drc_error_file` command supports only flat Cailbre DRC error files; it does not support Calibre hierarchy DRC error files.

You can load the Milkyway error cell file created by the `read_drc_error_file` command into the error browser to report or display the DRC violations. For information about using the error browser, see the following section, “Analyzing DRC Violations.”

Analyzing DRC Violations

After you have generated an error cell file, either by running one of the Synopsys DRC checking commands or by converting the Calibre results to an error cell file, you can analyze the DRC violations by

- Generating a report that lists the location of each error.

  To report the location of each violation, use the `report_error_coordinates` command.

- Viewing the errors in the error browser.

  You can select the errors you need to examine by error type or layer, view error information in the error browser, and view error locations in the layout view. You can filter the error list, mark errors as fixed, highlight errors in the layout view, and select errors interactively with the Error Selection tool. You can also save the errors in a file.

  For detailed information about using the error browser, see the “Examining Routing and Verification Errors” section in Appendix A of the *IC Compiler Implementation User Guide* and the “Examining Routing and Verification Errors” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

4

Using Zroute for Design for Manufacturing and Chip Finishing

This chapter describes how to use Zroute for design for manufacturing (DFM) and chip finishing tasks. It contains the following sections:

For information about additional chip finishing tasks, see the *IC Compiler Implementation User Guide*.

Reducing Via Count and Wire Length

Zroute optimizes via count and wire length during routing. There is no need to run a separate command to perform this optimization.

You can control the effort level used for via count and wire length optimization by setting the `optimize_wire_via_effort_level` detail route option. By default, low effort is used. If you are inserting redundant vias, use medium effort to reduce the initial via count. To further optimize via count and wire length, you can set the effort level to high.

Finding and Fixing Antenna Violations

The antenna flow consists of the following steps:

1. Define the antenna rules
2. Analyze and fix the antenna violations

The following sections describe these steps.

Defining Antenna Rules

You define the pin and port antenna properties either in a cell library format (CLF) file or by setting the following detail route options:

- `default_diode_protection`
- `default_gate_size`
- `default_port_external_gate_size`
- `default_port_external_antenna_area`
- `port_antenna_mode`

For information about setting detail route options, see “Setting Detail Route Options” on page 2-22.

If you are using a hierarchical flow and create an interface logic model (ILM) for a block, you must use the `extract_zrt_hier_antenna_property` command to extract the antenna information from the block to use at the next level of hierarchy.

You define the metal layer antenna rules by using the `define_antenna_rule` and `define_antenna_layer_rule` IC Compiler commands. For more information about these commands, see the chip finishing chapter of the *IC Compiler Implementation User Guide*.

Analyzing and Fixing Antenna Violations

If the Milkyway design library contains antenna rules, Zroute automatically analyzes and fixes antenna violations.

Just like other design rules, antenna rules are checked and corrected during detail routing. This concurrent antenna rule correction architecture reduces total runtime by minimizing the iterations.

By default, Zroute

- Starts fixing antenna violations in the second iteration, after initial routing is complete and the basic DRC violations have been fixed

  You can change the iteration in which Zroute starts fixing antenna violations by setting the `antenna_on_iteration` detail route option.

- Does not check or correct antenna rules for power and ground nets

  To check and correct antenna rules for power and ground nets, set the `check_antenna_on_pg` detail route option to true.

- Does not insert diodes to fix antenna violations

  To enable diode insertion, set the `insert_diodes_during_routing` detail route option to true. When true, Zroute inserts diodes to fix antenna violations when running detail routing. Zroute selects the diodes from the reference library and inserts them into existing open spaces. You can control which diodes are used by setting the `diode_libcell_names` detail route option. By default, Zroute reuses existing filler cell locations for diode insertion. You can prevent Zroute from reusing these locations by setting the `reuse_filler_locations_for_diodes` detail route option to false.

As with other design rule violations, antenna violations are reported at the end of each detail routing iteration.

For example,

```
DRC-SUMMARY:
   @@@@@@@ TOTAL VIOLATIONS =      506
    @@@@ Total number of instance ports with antenna violations =   1107
```

To disable the analysis and correction of antenna rules during detail routing, set the `antenna` detail route option to false.

```
icc_shell> set_route_zrt_detail_options -antenna false
```

To check for antenna violations, use the `verify_zrt_route -antenna` command. Due to the different connectivity models, do not use the `report_antenna_ratio` command on designs routed with Zroute. You can also run the `report_design -physical` command to report information about antenna violations.

Inserting Redundant Vias

You can perform redundant via insertion in the following ways:

- Postroute redundant via insertion
- Concurrent soft-rule-based redundant via insertion
- Concurrent hard-rule-based redundant via insertion

In general, you should start with postroute redundant via insertion. If doing so provides good results, you can use one of the concurrent redundant via insertion methods to improve the redundant via rate. If postroute redundant via insertion results in a redundant via rate of at least 80 percent, you can try to improve the redundant via rate by using concurrent soft-rule-based redundant via insertion. If postroute redundant via insertion results in a redundant via rate of at least 90 percent, you can try to improve the redundant via rate by using concurrent hard-rule-based redundant via insertion.

The following sections describe how to define the via mapping table, how to insert redundant vias by using the various methods, and how to report the redundant via rate.

Defining a Via Mapping Table

By default, Zroute reads the default contact codes from the technology file and generates an optimized via mapping table. To see the default via mapping table, use the `insert_zrt_redundant_vias -list_only` command (or choose Route > Insert Redundant Vias in the GUI and select “List redundant via mappings”).

Example 4-1 shows an example of a default via mapping table.

Example 4-1 Default Via Mapping Table

```
icc_shell > insert_zrt_redundant_vias -list_only
VIA12    -> VIA12T_1x2    VIA12T_2x1     VIA12_1x2     VIA12_2x1(r)
    VIA12_1x2(r)  VIA12_2x1

VIA12(r) -> VIA12T_1x2    VIA12T_2x1     VIA12_1x2     VIA12_2x1(r)
   VIA12_1x2(r)  VIA12_2x1

VIA23    -> VIA23T_2x1    VIA23T_1x2     VIA23_2x1     VIA23_2x1(r)
   VIA23_1x2     VIA23_1x2(r)

VIA23(r) -> VIA23T_2x1    VIA23T_1x2     VIA23_2x1     VIA23_2x1(r)
   VIA23_1x2     VIA23_1x2(r)
```

If you do not want to use this default mapping table, you can define a customized mapping table by using the `define_zrt_redundant_vias` command (or by choosing Route > Define Redundant Vias in the GUI).

By default, all mappings have the same priority, and Zroute selects the redundant vias to use based on routability. You can specify preferred redundant via mappings by using the `-to_via_weights` option to assign a weight value between 1 and 10 to each redundant via in the mapping table. During redundant via insertion, Zroute uses the higher weighted redundant vias first.

The syntax of the `define_zrt_redundant_vias` command is

```
define_zrt_redundant_vias
  [-from_via {list_of_from_vias}]
  [-to_via {list_of_to_vias}]
  [-to_via_x_size {list_of_number_of_contacts}]
  [-to_via_y_size {list_of_number_of_contacts}]
  [-to_via_weights {list_of_number_of_contacts}]
```

Note:
The via mapping table defined by the `define_zrt_redundant_vias` command is not saved in the Milkyway design library. You must define the via mapping table in each script or session in which you insert redundant vias.

Example 4-2 shows an example of using the `define_zrt_redundant_vias` command to define a customized via mapping table.

Example 4-2 Customized Via Mapping Table

```
icc_shell> define_zrt_redundant_vias \
   -from_via  { VIA12 VIA12 VIA23 VIA23 VIA34 VIA34 VIA45 VIA56S \
                VIA6STG VIATGAL VIA12E VIA12E VIA23E VIA23E VIA34E \
                VIA34E VIA45E VIA56SE VIA12T VIA12T VIA23T VIA23T \
                VIA34T VIA34T } \
   -to_via    { VIA12T VIA12 VIA23T VIA23 VIA34T VIA34 VIA45 VIA56S \
                VIA6STG VIATGAL VIA12T VIA12 VIA23T VIA23 VIA34T \
                VIA34 VIA45 VIA56S VIA12T VIA12 VIA23T VIA23 \
                VIA34T VIA34 } \
   -to_via_x_size  { 2 2 2 2 2 2 2 2 \
                     2 2 2 2 2 2 2 \
                     2 2 2 2 2 2 2 \
                     2 2 } \
   -to_via_y_size  { 1 1 1 1 1 1 1 1 \
                     1 1 1 1 1 1 1 \
                     1 1 1 1 1 1 1 \
                     1 1 } \
   -to_via_weights { 5 1 5 1 5 1 1 1 \
                     1 1 1 1 1 1 1 \
                     1 1 1 1 1 1 1 \
                     1 1 }
```

For more information about the `define_zrt_redundant_vias` command, see the man page.

Postroute Redundant Via Insertion

To perform postroute redundant via insertion, use the `insert_zrt_redundant_vias` command (or choose Route > Insert Redundant Vias in the GUI).

This command replaces single-cut vias with multiple-cut via arrays or with another single-cut via that has a different contact code. During redundant via insertion, the detail router also checks the design rules within the neighboring partition to minimize the DRC violations.

After all single vias are checked and replaced, the detail router rechecks for DRC violations and runs iterations to correct any violations. Set the number of detail routing iterations with the `max_number_iterations` detail route option. For information about setting the detail route options, see "Setting Detail Route Options" on page 2-22.

Note:
When you insert redundant vias in a multicorner-multimode design, Zroute uses only the current scenario.

After you perform the initial postroute redundant via insertion, set the `post_detail_route_redundant_via_insertion` common route option to enable automatic insertion of redundant vias after subsequent detail routing or ECO routing. This helps to maintain the redundant via rate in your design. For information about setting the common route options, see "Setting Common Route Options" on page 2-15.

You can also perform redundant via insertion during the `route_opt` flow by setting the `post_detail_route_redundant_via_insertion` common Zroute option before running `route_opt`.

If the percentage of redundant vias is not high enough, you can increase the effort level by using the `-effort` option to get a better redundant via rate. Increasing the effort level to high can increase the redundant via rate by about 3 to 5 percent by shifting the vias to make room for additional vias. However, because high-effort redundant via insertion moves the vias more, it can result in a less lithography-friendly pattern at the 45 nm technology node and below. In this case, you should use concurrent soft-rule-based redundant via insertion to improve the redundant via rate.

Concurrent Soft-Rule-Based Redundant Via Insertion

Soft-rule-based redundant via insertion can improve the redundant via rate by reserving space for the redundant vias during routing. You can use concurrent soft-rule-based redundant via insertion during both initial routing and ECO routing. The actual via insertion is not done during routing; you must still perform postroute redundant via insertion by using the `insert_zrt_redundant_vias` command.

Note:
Reserving space during routing increases the routing runtime. You should use this method only when needed to improve the redundant via rate beyond that provided by postroute redundant via insertion and the postroute approach resulted in a redundant via rate of at least 80 percent.

To perform concurrent soft-rule-based redundant via insertion,

1. (Optional) Define the via mapping table as described in "Defining a Via Mapping Table" on page 4-4.
2. Enable concurrent soft-rule-based redundant via insertion.

   By default, concurrent redundant via insertion is disabled.

   - To enable concurrent soft-rule-based redundant via insertion during initial routing, set the `concurrent_redundant_via_mode` common Zroute option to `reserve_space`.

     ```
     icc_shell> set_route_zrt_common_options \
     -concurrent_redundant_via_mode reserve_space
     ```

You can control the effort used to reserve space for the redundant vias during initial routing by setting the `concurrent_redundant_via_effort_level` common Zroute option.

Note:
If you enable the `concurrent_redundant_via_mode` option before running the `place_opt -congestion` command, the redundant vias are considered during congestion estimation.

- To enable concurrent soft-rule-based redundant via insertion during ECO routing, set the `eco_route_concurrent_redundant_via_mode` common Zroute option to `reserve_space`.

  ```
  icc_shell> set_route_zrt_common_options \
  -eco_route_concurrent_redundant_via_mode reserve_space
  ```

  You can control the effort used to reserve space for the redundant vias during ECO routing by setting the `eco_route_concurrent_redundant_via_effort_level` common Zroute option.

3. Route the design.

   During routing, Zroute reserves space for the redundant vias and fixes hard design rule violations.

4. Perform postroute redundant via insertion.

   During postroute redundant via insertion, Zroute inserts the redundant vias in the reserved locations.

Concurrent Hard-Rule-Based Redundant Via Insertion

Hard-rule-based redundant via insertion can improve the redundant via rate by treating redundant vias as hard design rules during routing. You can use concurrent hard-rule-based redundant via insertion only during initial routing; this method is not supported during ECO routing.

Note:
This method can result in a very large runtime increase for congested designs. You should use this method only when needed to improve the redundant via rate beyond that provided by concurrent soft-rule-based redundant via insertion and the soft-rule-based approach resulted in a redundant via rate of at least 90 percent.

To perform concurrent hard-rule-based redundant via insertion,

1. (Optional) Define the via mapping table as described in "Defining a Via Mapping Table" on page 4-4.
2. Enable concurrent hard-rule-based redundant via insertion by setting the `concurrent_redundant_via_mode` common Zroute option to `insert_at_high_cost`. (By default, concurrent redundant via insertion is disabled.)

   ```
   icc_shell> set_route_zrt_common_options \
   -concurrent_redundant_via_mode insert_at_high_cost
   ```

   You can control the effort used to reserve space for the redundant vias by setting the `concurrent_redundant_via_effort_level` common Zroute option.

   Note:
   If you enable the `concurrent_redundant_via_mode` option before running the `place_opt -congestion` command, the redundant vias are considered during congestion estimation.
3. Route the design.

   During routing, Zroute inserts the redundant vias and fixes hard design rule violations. In general, redundant via insertion has the same priority as other hard design rules; however, if the design rule violations do not converge during detail routing, Zroute automatically relaxes the redundant via constraints to ensure a DRC-clean result.

Preserving Timing During Redundant Via Insertion

When you insert redundant vias, it changes the timing of your design. Short nets tend to slow down due to increased capacitance, whereas long nets tend to speed up due to decreased resistance. Zroute redundant via insertion has a timing-preservation mode that allows you to perform redundant via insertion without impacting the design timing by preventing insertion of redundant vias on critical nets.

To enable timing-preservation mode for redundant via insertion, define the timing preservation constraints by using the following options of the `insert_zrt_redundant_via` command:

- `-timing_preserve_setup_slack_threshold`
- `-timing_preserve_hold_slack_threshold`
- `-timing_preserve_nets`

Use these options to increase the double via rate only after timing is met or almost met, because these options can severely reduce the redundant via rate due to critical nets.

Maximizing the Redundant Via Rate

To maximize the redundant-via insertion rate, use the following flow to insert redundant vias:

1. (Optional) Define the via mapping (`define_zrt_redundant_vias`).
2. Route the design (`route_opt -intital_route_only`).
3. Insert redundant vias using the default mode (`insert_zrt_redundant_vias`).
4. Enable concurrent redundant via insertion (`set_route_zrt_common -eco_route_concurrent_redundant_via_mode reserve_space -post_detail_route_redundant_via_insertion medium`).
5. Perform postroute optimization (`route_opt -skip_initial_route`).
6. Perform DFM tasks.
7. Perform incremental postroute optimization (`route_opt -incremental -size_only`)
8. Insert redundant vias using timing-preservation mode.

Reporting Redundant Via Rates

After redundant via insertion, whether concurrent or postroute, Zroute generates a redundant via report that provides the following information:

- The optimized via conversion rate for each layer

  The optimized via conversion rate includes both double vias and DFM-friendly bar vias, which have a single cut but a larger metal enclosure.
- The distribution of optimized vias by weight for each layer

  For information about assigning weights to redundant via mappings, see “Defining a Via Mapping Table” on page 4-4.
- The overall double via conversion rate for the design

Example 4-3 shows an example of the redundant via report.

Example 4-3 Redundant Via Report

```
Layer V01        = 41.89% (490617 / 1171301 vias)
    Weight 10    =  9.64% (112869  vias)
    Weight 5     = 32.25% (377689  vias)
    Weight 1     =  0.01% (59      vias)
    Un-optimized = 58.11% (680684  vias)
Layer V02        = 76.20% (1567822/ 2057614 vias)
    Weight 10    = 43.51% (895270  vias)
    Weight 5     = 28.62% (588805  vias)
    Weight 1     =  4.07% (83747   vias)
    Un-optimized = 23.80% (489792  vias)
Layer V03        = 81.87% (687115 / 839297  vias)
    Weight 10    = 64.50% (541369  vias)
    Weight 5     = 10.75% (90224   vias)
    Weight 1     =  6.62% (55522   vias)
    Un-optimized = 18.13% (152182  vias)
Layer V04        = 81.60% (226833 / 277977  vias)
    Weight 10    = 81.45% (226418  vias)
    Weight 1     =  0.15% (415     vias)
    Un-optimized = 18.40% (51144   vias)
Layer V05        = 89.12% (122470 / 137425  vias)
    Weight 10    = 89.12% (122470  vias)
    Un-optimized = 10.88% (14955   vias)
Layer V06        = 93.04% (77728  / 83547   vias)
    Weight 10    = 93.02% (77715   vias)
    Weight 1     =  0.02% (13      vias)
    Un-optimized =  6.96% (5819    vias)
Layer V07        = 93.71% (37448  / 39962   vias)
    Weight 10    = 93.71% (37448   vias)
    Un-optimized =  6.29% (2514    vias)
Layer V08        = 97.54% (13179  / 13511   vias)
    Weight 10    = 97.54% (13179   vias)
    Un-optimized =  2.46% (332     vias)
Layer V09        = 85.47% (1329   / 1555    vias)
    Weight 10    = 85.47% (1329    vias)
    Un-optimized = 14.53% (226     vias)

Total double via conversion rate    = 46.69% (2158006 / 4622189 vias)
```

You can also use the `report_design -physical` command to report the double via rate. This command reports the double via rate per layer, but does not provide any weighting information.

Reducing Critical Areas

A critical area is a region of the design where, if the center of a random particle defect falls there, the defect will cause circuit failure, thereby reducing yield. A conductive defect causes a short fault, and a nonconductive defect causes an open fault.

The following sections describe how to

- Report critical areas
- Display critical area maps
- Reduce critical area short faults by performing wire spreading
- Reduce critical area open faults by performing wire widening

Reporting Critical Areas

After routing is complete, you can report layout-critical areas that are susceptible to random particle defects, which cause shorts and opens, during the fabrication process.

The results from the critical area analysis report are output to an output_heatmap text file. You can see the critical area results graphically by displaying critical area heat maps.

To report critical areas, use the `report_critical_area` command (or choose Finishing > Report Critical Area Map in the GUI).

Table 4-1 describes the `report_critical_area` options. For more information, see the man page.

Table 4-1 report_critical_area Command Options

Command option	GUI object	Description
`-fault_type short | open`	"Fault type" radio buttons	Specifies the type of fault to check for. The default is `short`.
`-particle_distr_func_file` *`filename`*	"Particle distribution function file" box	The name of the input file that contains the particle distribution function to be used during the calculation. By default, Zroute uses an internal particle probability function.

Table 4-1 report_critical_area Command Options (Continued)

Command option	GUI object	Description
`-suppress_zeros_in_report`	"Suppress zeros in report" check box	Prevents reporting of zero results. By default, all results are reported.
`-multiparticle_report_format true | false`	"Multi particle report format" check box	Outputs a report containing the critical area analysis result of each individual particle size. By default, Zroute outputs normalized results.
`-tsmc_encr_particle_distr_file`	"TSMC encrypted format" check box	Specifies that the particle distribution function file is in TSMC encrypted format.
`-layer_alias_DSD_format`	"Metal scheme directives" boxes	Specifies the layer alias used for defect size distribution (DSD) of each layer.
`-input_layers`	"Input layer" list	Specifies the metal layers for which the critical area is calculated. By default, the critical area is calculated for all metal layers.

Often the particle probability function is considered sensitive data. When security for a sensitive particle distribution file from a foundry is of concern, use the `process_particle_probability_file` command, which provides a way to encrypt and decrypt the particle probability function. Critical area analysis can work with such an encrypted particle probability function.

A secret key is used to encrypt the particle probability function. The encrypted file cannot be decrypted without the key. To encrypt and decrypt a particle probability function file, use the `process_particle_probability_file` command. The syntax of this command is

```
process_particle_probability_file
  -key string
  -input_file filename
  -output_file filename
```

Critical area analysis takes the encrypted file as its input and processes it without the key. The output will be the heat map based on the encrypted particle probability function; it will be in text format. The text format of the particle probability function is still accepted as input to critical area analysis. If an encrypted file is given as input, the file is internally decrypted and used.

Critical area analysis takes the encrypted file as its input and processes it without the key. The output will be the heat map based on the encrypted particle probability function; it will be in text format. The text format of the particle probability function is still accepted as input to critical area analysis. If an encrypted file is given as input, the file is internally decrypted and used.

Displaying Critical Area Maps

Critical area maps provide an indication of places where a chip might fail due to particle defects, causing shorts and opens.

To display a critical area map for the current design,

1. Specify the type of defect to display by choosing one of the following:
   - Finishing > Short Critical Area Map
   - Finishing > Open Critical Area Map

   The Map Mode panel appears.
2. Select a metal layer for which critical area shorts or opens are to be displayed.
3. Modify the critical heat ranges by changing (or removing) the maximum or minimum threshold values.
4. Adjust other display parameters. You can make text visible in the map.
5. Click Apply.

   To update the critical area map data after you perform wire spreading, click Reload. This action opens the (Re)Calculate Short Critical Area Map Data dialog box, from which you can run the `report_critical_area -fault_type short` command.

   To update the critical area map data after you perform wire widening, click Reload. This action opens the (Re)Calculate Open Critical Area Map Data dialog box, from which you can run the `report_critical_area -fault_type open` command.

For more information about using critical area maps, see the “Examining Critical Area Maps” topic in IC Compiler Help.

Performing Wire Spreading

After you have performed detail routing and redundant via insertion, you can perform wire spreading to increase the average spacing between wires, which reduces the critical area short faults and therefore improves yield.

To perform wire spreading, use the `spread_zrt_wires` command (or choose Finishing > Route Spread Wires in the GUI).

By default, the `spread_zrt_wires` command spreads the signal wires on the same layer by half a pitch in the preferred direction. You can change the spread distance by using the `-pitch` option. After spreading, the `spread_zrt_wires` command performs detail routing iterations to fix any DRC violations caused as a result of spreading.

When you change the layout, it can change the timing of your design. Zroute wire spreading has a timing-preservation mode that allows you to perform wire spreading without impacting the design timing.

To enable timing-preservation mode for wire spreading, define the timing preservation constraints by using the following options of the `spread_zrt_wires` command:

- `-timing_preserve_setup_slack_threshold` *`threshold`*
- `-timing_preserve_hold_slack_threshold` *`threshold`*
- `-timing_preserve_nets` *`nets`*

The threshold values are floating point numbers in library units. Wire spreading is not performed on nets with slack less than the specified values or those specified in the `-timing_preserve_nets` option.

Performing Wire Widening

After you have performed detail routing, redundant via insertion, and wire spreading, you can perform wire widening to increase the average width of the wires, which reduces the critical area open faults and therefore improves yield.

To perform wire widening, use the `widen_zrt_wires` command (or choose Finishing > Route Widen Wires in the GUI).

The `widen_zrt_wires` command widens all wires in the design to 1.5 times their original width and then performs detail routing iterations to fix any DRC violations caused as a result of widening. Note that the widened wires do not trigger fat wire spacing rules.

When you widen the wires, it changes the timing of your design. Zroute wire widening has a timing-preservation mode that allows you to perform wire widening without impacting the design timing.

To enable timing-preservation mode for wire widening, define the timing preservation constraints by using the following options of the `widen_zrt_wires` command:

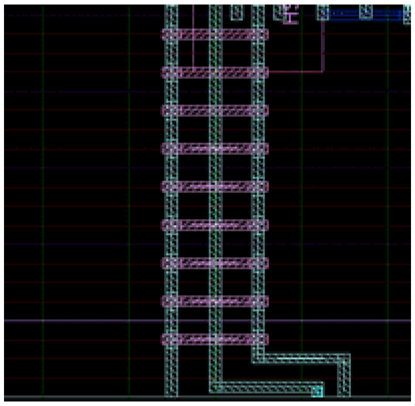
- `-timing_preserve_setup_slack_threshold` *threshold*
- `-timing_preserve_hold_slack_threshold` *threshold*
- `-timing_preserve_nets` *nets*

The threshold values are floating point numbers in library units. Wire widening is not performed on nets with slack less than the specified values or those specified in the `-timing_preserve_nets` option.

Shielding Nets

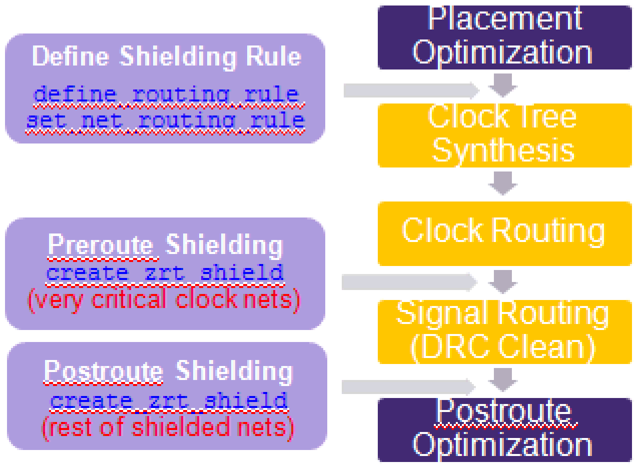
The router shields routed nets by generating shielding wires that are based on the shielding widths and spacing defined in the shielding rules. In addition to shielding nets on the same layer, you also have the option to shield one layer above or one layer below or the layer above and the layer below. Shielding above or below the layer is called *coaxial shielding*. Coaxial shielding provides even better signal insulation than same-layer shielding, but it uses more routing resources.

Figure 4-1 Coaxial Shielding



You can perform shielding either before or after signal routing. Shielding before signal routing, which is referred to as preroute shielding, provides better shielding coverage, but can result in congestion issues during signal routing. Preroute shielding is typically used to shield critical clock nets. Shielding after signal routing, which is referred to as postroute shielding, has a very minimal impact on routability, but provides less protection to the shielded nets. Figure 4-2 shows the Zroute shielding flow, which is described in the sections that follow.

Figure 4-2 Zroute Shielding Flow



Defining the Shielding Rules

Before you perform shielding, you must

1. Define the shielding rules.

   To define shielding rules, use the `-shield_spacings` and `-shield_widths` options of the `define_routing_rule` command. For example, to specify a shielding rule that uses spacing of 0.1 microns and width of 0.1 microns for metal1 through metal5 and spacing of 0.3 microns and width of 0.3 microns for metal6, enter the following command:

```
icc_shell> define_routing_rule shield_rule \
-shield_widths {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3} \
-shield_spacing {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3}
```

   For more information about the `define_routing_rule` command, see "Defining Nondefault Routing Rules" on page 2-9.

2. Assign shielding rules to the nets to be shielded.

   To avoid congestion issues and achieve the best balance of DRC convergence and timing closure, you should apply shielding rules only to high-frequency or critical clock nets and apply double-spacing rules to the lower-frequency clock nets.

   Use the `set_clock_tree_options` command to assign shielding rules to clock nets. For signal nets, use the `set_net_routing_rule` command to assign the shielding rules.

Note:

The `set_clock_tree_options` command assigns shielding rules to clock nets only before clock tree synthesis. After clock tree synthesis, you must use the `set_net_routing_rule` command to assign shielding rules to clock nets.

For more information about these commands, see "Applying Nondefault Routing Rules" on page 2-10.

Performing Preroute Shielding

To provide the most protection for critical clock nets, perform shielding on those nets after clock tree routing but before signal net routing.

To add shielding to the routed clock nets based on the assigned shielding rules, use the `create_zrt_shield` command (or choose Route > Create Shield in the GUI).

By default, the `create_zrt_shield` command performs same-layer shielding on all nets with predefined shielding rules. The shielding wires are tied to ground, standard cell power and ground pins, and standard cell rails.

To explicitly specify the nets on which to perform shielding, use the `-nets` option. To tie the shielding wires to a named power or ground net, use the `-with_ground` option. To prevent connections to the standard cell power and ground pins, use the `-ignore_shielding_net_pins` option.

To perform coaxial shielding, use the `-coaxial_above` and `-coaxial_below` options. By default, the `create_zrt_shield` command leaves one routing track open between each used track. For coaxial shielding above the shielded net segment layer (`-coaxial_above true`), you control the number of open tracks between used tracks by using the `-coaxial_above_skip_tracks` option. For coaxial shielding below the shielded net segment layer (`-coaxial_below true`), you control the number of open tracks between used tracks by using the `-coaxial_below_skip_tracks` option. For either of these options, you can specify a value between zero and seven.

For example, to perform coaxial shielding below the shielded net segment layer on the clock nets, prevent signal routing below the shielded net segment layer, and tie the shielding wires to VSS, enter

```
icc_shell> create_zrt_shield -nets $clock_nets \
-coaxial_below true -coaxial_below_skip_tracks 0 \
-with_ground VSS
```

When you run the `create_zrt_shield` command, it reports the shielding rate achieved for each net, as shown in the following example:

```
Shielded 100% side-wall of (CLK_G1B2I2)
Shielded 100% side-wall of (CLK_G1B2I3)
Shielded 100% side-wall of (CLK_G1B2I6)
Shielded 100% side-wall of (CLK_G1B2I8)
Shielded 100% side-wall of (CLK_G1B2I4)
Shielded 100% side-wall of (CLK_G1B2I7)
Shielded 100% side-wall of (CLK_G1B1I1)
Shielded 100% side-wall of (CLK_G1B1I2)
Shielded 100% side-wall of (CLK)
Shielded 204 nets with average ration of 100.00%)
```

Soft Shielding Rules During Signal Routing

Zroute considers shielding rules as soft rules during signal routing. If a net has a shielding rule and is not shielded before signal routing, Zroute reserves shielding space during the whole routing process, global routing, track assignment, and detail routing. At the end of detail routing, it reports the shielding space violations and the locations where shielding wires cannot be established. The following log file example shows shielding soft spacing violations, which are highlighted in bold text:

```
DRC-SUMMARY:
        @@@@@@@ TOTAL VIOLATIONS = 13
        Diff net var rule spacing : 2
        Same net spacing : 2
        Less than minimum area : 4
        Short : 1
        Soft spacing (shielding) : 2
```

Signal routing only reserves space for the shielding; it does not actually insert it. You must run `create_zrt_shield` after signal routing to physically place the shielding wires. The reported soft rule violations help you to understand the shielding rate. Note that the router reserves space only for same-layer shielding and not for coaxial shielding; therefore, postroute coaxial shielding can produce a very low shielding rate.

Performing Postroute Shielding

To perform postroute shielding, you use the same command, `create_zrt_shield`, that is used for preroute shielding.

If you want to use the default spacings and widths from the technology file for postroute shielding, you do not need to define and assign nondefault routing rules. If you specify the nets to be shielded by using the `-nets` option, `create_zrt_shield` shields these nets with the default spacing and widths.

Postroute shielding should introduce few to no DRC violations. If DRC violations are created during shielding, `create_zrt_shield` triggers five detail routing iterations to fix them. If this does not fix the DRC violations, you can fix the remaining violations by running incremental detail routing with the `route_zrt_detail -incremental` command. It is possible that postroute shielding might break some tie-off connections during the shield trimming process. In this case, use `route_zrt_eco` instead of `route_zrt_detail` to rebuild the tie-off connections and to fix the DRC violations.

Shielding Example

Example 4-4 provides an example of shielding clock nets using preroute shielding and shielding critical nets using postroute shielding.

Example 4-4 Shielding Flow Example

```
# Define shielding rule
define_routing_rule shield_rule \
  -shield_widths {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3} \
  -shield_spacing {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3}

# Assign shielding rule to clock nets
set_clock_tree_options -clock_tree CLK \
  -routing_rule shield_rule

# Perform clock tree synthesis
set_delay_calculation -clock_arnoldi
clock_opt -no_clock_route

# Route the clock nets, reusing the global routing result from clock_opt
route_zrt_group -all_clock_nets -reuse_existing_global_route true

# Perform preroute shielding for the clock nets
create_zrt_shield -nets $clock_nets \
  -with_ground VSS

# Assign shielding rule to critical nets
set_net_routing_rule -rule shield_rule $critical_nets

# Route signal nets using shielding soft rules
route_zrt_auto

# Perform postroute shielding for critical nets
create_zrt_shield -nets $critical_nets \
  -with_ground VSS
```

Inserting Filler Cells

You can fill empty spaces in the standard cell rows with instances of reference filler cells to make sure all power nets are connected. One method of improving the stability of the power supply is to add decoupling capacitors as filler cells.

You insert filler cells by using the `insert_stdcell_filler` command (or by choosing Finishing > Insert Standard Cell Filler in the GUI). Zroute supports filler cells with and without metal and supports both single-height and multiheight filler cells.

To insert filler cells using Zroute,

1. Enable Zroute as the router by setting the Zroute route mode option to true.

   ```
   icc_shell> set_route_mode_options -zroute true
   ```

2. (Optional) Define the standard cell filler rules by using the `set_left_right_filler_rule` command to define the left and right filler rules. These rules specify the filler cell to insert immediately to the left and right, respectively, of specific standard cells.
3. Insert filler cells with metal.

   When you insert filler cells with metal, you must use the `-cell_with_metal` option to specify the reference filler cells and the `-connect_to_power` and `-connect_to_ground` options to connect the inserted filler cells to power and ground.

   Zroute checks for DRC violations when you insert filler cells with metal.

   - If you do not specify the `-connect_to_power` and `-connect_to_ground` options, Zroute might see mismatched power and ground errors and remove the filler cells when checking for shorts.
   - If a filler cell with metal causes DRC violations, Zroute removes it.
4. Insert filler cells without metal.

   When you insert filler cells without metal, you must use the `-cell_without_metal` option to specify the reference filler cells. You can make the power and ground connections to these filler cells either by using the `-connect_to_power` and `-connect_to_ground` options when you insert them or by using the `derive_pg_connection` command after inserting them.

   Zroute does not check for DRC violations when you insert filler cells without metal. To remove filler cells without metal that cause DRC violations, use the `remove_zrt_filler_with_violation` command. Because removing the filler cells can expose new violations, you sometimes need to run this command multiple times to remove all violating filler cells.

For example, to insert filler cells and connect them to the VDD power net and VSS ground net, enter the following commands:

```
icc_shell> insert_stdcell_filler -cell_with_metal $FILLER_CELL_METAL \
-connect_to_power VDD -connect_to_ground VSS
icc_shell> insert_stdcell_filler -cell_without_metal $FILLER_CELL \
-connect_to_power VDD -connect_to_ground VSS
```

For more information about these steps, see the "Chip Finishing and Design for Manufacturing" chapter in the *IC Compiler Implementation User Guide*.

Inserting Metal Fill

After routing, you can fill the empty tracks in the design with metal wires to meet the metal density rules required by most fabrication processes. This should be done before layout versus schematic (LVS) connectivity verification, design rule checking (DRC), and layout parasitic extraction.

When you define minimum and maximum metal density rules in the technology file, IC Compiler tries to create fill within the specified ranges.

The router does not recognize dummy metal that the GDS reader creates in the FILL view because the dummy metal areas are created as polygons, not wires. However, if they are converted to rectangles and marked with the Fill Track route type attribute, the router will recognize them correctly. Use the `convert_fill_polygons` command to convert the existing FILL view polygons to rectangles and to mark the rectangles with the Fill Track route type. This command should be called as part of data preparation so that the router does not have a problem reading the fill wires. The syntax is

```
convert_fill_polygons -library library_name -from cell_name -to cell_name
```

where *library_name* is the name of the Milkyway library containing the design cell to modify and *cell_name* is the name of the design cell to modify.

The `convert_fill_polygons` command converts the input cell's FILL view polygons to rectangles and then stores the cell in the FILL view of the specified output cell name. The input cell is not actually modified unless the output cell name is the same as the input cell name.

IC Compiler supports both real and emulation metal fill extraction.

- To perform emulation metal fill extraction, you must specify the emulation TLUPlus files by using the `-max_emulation_tluplus` and `-min_emulation_tluplus` options of the `set_tlu_plus_files` command.
- To perform real metal fill extraction, you must specify the `-real_metalfill_extraction` option of the `set_extraction_options` command. IC Compiler can treat the metal fill polygons as either floating or grounded. You can either specify how to treat the metal fill polygons (use the `floating` or `grounded` keyword) or let IC Compiler determine the treatment, based on the fill wire track property (use the `auto` keyword). When you use the `none` keyword (the default), fill is not considered during extraction.

  In addition to specifying the `-real_metalfill_extraction` option, you must specify TLUPlus files without any emulation information, using the `set_tlu_plus_files` command.

  Note:
  IC Compiler uses non-emulation TLUPlus files when `set_extraction_options -real_metalfill_extraction` is set to `auto`, `floating`, or `grounded`. Therefore, you should select real metal fill for extraction only after the metal fill objects have been added.

There are two ways to insert metal fill: by using the internal IC Compiler metal fill capability (`insert_metal_filler` command or Finishing > Insert Metal Filler in the GUI) or by invoking the external Hercules metal fill capability (`signoff_metal_fill` or Finishing > Signoff Metal Fill in the GUI). Using Hercules metal fill ensures DRC conformance and produces better quality of results, but requires a Hercules license.

Note:
When you use the `insert_metal_filler` command on a design that is routed with Zroute, you cannot use the `-tie_to_net ground` option, due to the connectivity differences between Zroute and the classic router.

For more information about inserting metal fill, see the “Chip Finishing and Design for Manufacturing” chapter in the *IC Compiler Implementation User Guide*.

Index